NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

RADIO IMPLEMENTATION OF A TESTBED FOR COGNITIVE RADIO SOURCE LOCALIZATION USING USRPs AND GNU RADIO

by

Amir Jerbi

September 2014

Thesis Advisor:	Murali Tummala
Co-Advisor:	John McEachen

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2014	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE RADIO IMPLEMENTATION OF A TESTBED FOR COGNITIVE RADIO SOURCE LOCALIZATION USING USRPs AND GNU RADIO			**5. FUNDING NUMBERS**
6. AUTHOR(S) Amir Jerbi			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The shift from wired to fully wireless communication is causing an increasing demand on the frequency spectrum. The cognitive radio was introduced to solve spectrum scarcity by allowing spectrum sharing between licensed and unlicensed users. This approach presents a challenge to source localization because of the cognitive radio's capability to shift its spatial, frequency and temporal parameters. The extended semi-range-based (ESRB) and cooperative-received-signal-strength-based (CRSSB) localization schemes are proposed to overcome the challenge of identifying and locating a cognitive radio over time using a wireless sensor network. The objective of this thesis was to set up a testbed using GNU Radio and Universal Software Radio Peripherals (USRPs) to estimate the position of a cognitive radio device using the ESRB and CRSSB localization schemes. The ESRB algorithm does not provide accurate position estimates but the estimates are observed to be concentrated in the vicinity and converging toward the true position of the secondary user. The errors are believed to be caused by three factors: a limited number of sensor nodes used (four), an insufficient number of spectral scans per superframe (55), and the lack of synchronization among sensor nodes. The CRSSB localization scheme gave a more accurate position estimation.

14. SUBJECT TERMS cognitive radio, source localization, extended semi range-based localization, cooperative received signal strength based localization, wireless sensor networking			**15. NUMBER OF PAGES** 109
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

RADIO IMPLEMENTATION OF A TESTBED FOR COGNITIVE RADIO SOURCE LOCALIZATION USING USRPs AND GNU RADIO

Amir Jerbi
Captain, Tunisian Air Force
EE, EABA, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2014

Author:	Amir Jerbi
Approved by:	Murali Tummala Thesis Advisor
	John McEachen Co-Advisor
	R. Clark Robertson, Ph.D. Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The shift from wired to fully wireless communication is causing an increasing demand on the frequency spectrum. The cognitive radio was introduced to solve spectrum scarcity by allowing spectrum sharing between licensed and unlicensed users. This approach presents a challenge to source localization because of the cognitive radio's capability to shift its spatial, frequency and temporal parameters. The extended semi-range-based (ESRB) and cooperative-received-signal-strength-based (CRSSB) localization schemes are proposed to overcome the challenge of identifying and locating a cognitive radio over time using a wireless sensor network. The objective of this thesis was to set up a testbed using GNU Radio and Universal Software Radio Peripherals (USRPs) to estimate the position of a cognitive radio device using the ESRB and CRSSB localization schemes. The ESRB algorithm does not provide accurate position estimates but the estimates are observed to be concentrated in the vicinity and converging toward the true position of the secondary user. The errors are believed to be caused by three factors: a limited number of sensor nodes used (four), an insufficient number of spectral scans per superframe (55), and the lack of synchronization among sensor nodes. The CRSSB localization scheme gave a more accurate position estimation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ς	cyclic frequency
b	frequency step
β	detection threshold
E_{av}	estimated average energy
E_{rv}	average received energy
F	FFT vector size
F_S	sampling rate
g	gain
h	channel attenuation
H_0	hypothesis 0
H_1	hypothesis 1
I	in-phase
L	packet size
N	number of transmitted bursts
N_{HE}	the number of times the estimated average energy is higher than a preselected threshold
N_T	total number of iterations
P_f	probability of false alarm
$P_n(t)$	noise level in the surrounding environment
$P_s(t)$	transmitted signal power
$P_x(t)$	power of the received signal
Q	quadrature
R_x^{ς}	cyclic autocorrelation function
t	time
T	time interval
T_{SF}	burst duration
x	detected signal
c2magsq	complex 2 magnitude squared
s2v	bit stream 2 victor
ADC	Analog-to-Digital Converter

CRSSB	Cooperative-Received-Signal-Strength-Based
DAC	Digital-to-Analog Converter
DC	Direct Current
DOD	Department of Defense
ESRB	Extended Semi-Range-Based
FCC	Federal Communications Commission
FDMA	Frequency-Division Multiple Access
FFT	Fast Fourier Transformation
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific and Medical
PSK	Phase-Shift Keying
PU	Primary User
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
R&D	Research and Development
RF	Radio Frequency
SNR	Signal-to-Noise Ratio
SU	Secondary User
SWIG	Simplified Wrapper and Interface Generator
thres	Threshold
TMoD	Tunisian Ministry of Defense
USR	Ultimate Software Radio
USRP	Universal Software Radio Peripheral
WRAN	Wireless Regional Area Network

EXECUTIVE SUMMARY

Communication is shifting from wired to a fully wireless technology, causing an increasing demand for radio frequency spectrum leading to a shortage in available frequency bands. Nevertheless, by observing the radio spectrum over time, it can be seen that some radio frequency bands are heavily used, especially the unlicensed bands, whereas some licensed bands are underutilized and only partially occupied.

Cognitive radio was introduced as a solution to improve spectrum utilization by allowing spectrum sharing between licensed and unlicensed users. The cognitive radio is an intelligent device with the capability of detecting the surrounding spectrum occupancy and selecting the suitable parameters (e.g., frequency and modulation) to opportunistically access the spectrum without affecting the quality of the licensed user's communication.

Due to the high demand of wireless devices and the shortage of the frequency spectrum, both the U.S. Department of Defense (DOD) and the Tunisian Ministry of Defense (TMoD) are moving toward a heavy use of cognitive radio technologies in their wireless communication. It is challenging for any military application to locate deployed cognitive radios in the area of operation for two reasons. First, any localization scheme must be able to adapt along with the cognitive radio as it changes. Second, the scheme requires keeping track of the cognitive radio's frequency occupancy to distinguish between licensed users and cognitive radios.

Angle-of-arrival and received-signal-strength-based localization are two localization algorithms that are commonly used in a cognitive environment. The accuracy of these schemes requires a precise channel model and *a priori* knowledge of the transmission conditions (e.g., signal-to-noise ratio and path loss factor). The cooperative-received-signal-strength-based localization scheme (CRSSB) is capable of solving for the position of a secondary user in cognitive environment using a wireless sensor network without the prior knowledge of transmission conditions.

An extended semi-range-based (ESRB) location scheme is another scheme that has been proposed to overcome the challenge of identifying and tracking the position of a cognitive radio over time. The scheme's underlying principle is the monitoring of the environment's temporal parameters (i.e., position and frequency occupancy) in a collaborative manner to determine the cognitive radio's position.

The objective of this thesis was to implement a real-world software-defined radio environment experiment in which the position of a cognitive radio device was estimated using the ESRB and CRSSB localization schemes. The network elements were designed based upon the software-defined radio approach, using a GNU Radio interfaced with Ettus Research's Universal Software Radio Peripheral (USRP) devices. Three GNU Radio routines were developed to meet the design requirements of the sensor node, the primary user, and the secondary user. Two available devices from Ettus Research were used: the USRP N210 with WBX daughterboard for sensor nodes and the secondary user (cognitive radio device) and the USRP B200 for primary users.

The cognitive environment experimental testbed was set up on the roof of Spanagel Hall at the Naval Postgraduate School. Each of the networked elements worked successfully and provided the desired output. First, the primary user generated a signal with fixed amplitude at the preselected channel. Second, all sensor nodes were able to perform the energy detection process of the primary user signal. Finally, the secondary user was able to sense the spectrum and transmit a burst in the detected vacant slots.

As a final step, the scan reports from each sensor node were aggregated at the decision maker in which the ESRB and the CRSSB localization algorithms were executed to estimate the secondary user location. For the ESRB localization scheme, the results were not accurate, but the estimates are observed to be concentrated in the vicinity and converging toward the true position of the secondary user. The position errors are believed to be caused by three factors: a limited number of the sensor nodes used (four sensor nodes), a number of spectral scans per superframe (55 scans) which were fewer than the suggested number to obtain close estimates (600 scans), and a lack of timing synchronization among sensor nodes. The CRSSB localization scheme provided position estimation within an acceptable level of tolerance.

ACKNOWLEDGMENTS

I would like to offer my gratitude to Professor Murali Tummala, Professor John McEachen, Robert Broadston, and Donna Miller of the Naval Postgraduate School; Major Agur Adams USMC; and Lieutenant Carson McAbee USN, for their invaluable contribution to this work.

I would like also to thank writing coach Chloe Woida of the NPS Graduate Writing Center, for her precious help in writing this thesis.

Finally, special thanks to my mother, father, and wife for their support, prayers, patience, and thoughts throughout my time at NPS.

To Falfoula, Dad loves you and I am sorry for being away for two years.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Currently, communication is switching from wired to a fully wireless technology. Moreover, the demand for wireless applications is expanding, causing an increasing demand for radio frequency spectrum [1], [2], [3]. To establish a beneficial use of the radio spectrum, the Federal Communication Commission (FCC) in the United States and similar governmental agencies in other countries, are regulating frequency spectrum access between users by assigning frequency bands to specific users (licensed users) in a specific location.

The FCC is facing the challenge of finding free frequency slots for new services which is considered the hardest problem to solve because of spectrum scarcity. Nevertheless, by observing the radio spectrum over time, it can be seen that some radio frequency bands are heavily used, especially the unlicensed bands, whereas some licensed bands are underutilized and only partially occupied [2], [4], [5].

Cognitive radio was introduced as a solution to improve spectrum utilization by allowing spectrum sharing between licensed and unlicensed users. A cognitive radio is an intelligent device with the capability of being aware of the radio frequency occupancy and selecting the suitable parameters (e.g., frequency and modulation) to opportunistically access the spectrum without affecting the licensed user's communication quality [1], [2], [6], [7], [8].

Both the U.S. Department of Defense (DOD) and the Tunisian Ministry of Defense (TMoD) are moving toward a heavy use of cognitive radio technologies in their wireless communication due to high demand on wireless devices and the shortage of frequency spectrum. It is always important for any military application to be aware of the location of any deployed wireless device in the area of operation, which is challenging when considering these cognitive radio devices for two reasons. First, any localization scheme must be able to adapt along with the cognitive radio as it changes. Second, the scheme requires keeping track of the cognitive radio's frequency occupancy to distinguish between licensed users and cognitive radios [1], [2].

Angle-of-arrival and received-signal-strength-based-localization are two localization algorithms that are commonly used in cognitive environments. The accuracy of these schemes requires a precise channel model and *a priori* knowledge of the transmission conditions such as signal-to-noise ratio and path loss factor [9]. The cooperative-received-signal-strength-based localization scheme (CRSSB) was proposed in [9] to determine if it is possible to solve for the position of the secondary user in a cognitive environment using a wireless sensor network.

An extended semi-range-based (ESRB) location scheme is proposed in [1], [2] to overcome the challenge of identifying and tracking the position of a cognitive radio over time. The scheme's underlying principle is the monitoring of the environment's temporal parameters (i.e., position and frequency occupancy) in a collaborative manner to determine the cognitive radio's position [1], [2]. In order to test the feasibility and the efficacy of both schemes (ESRB and CRSSB localization) in real word conditions and to demonstrate that a wireless sensor network can be used to locate a cognitive radio over time, a scenario is implemented using software defined radios in this work.

A. THESIS OBJECTIVE

The objective of this thesis is to implement a real-world testing environment in which the position of a cognitive radio device is estimated using the ESRB and CRSSB localization schemes. To take advantage of software defined radio features (mainly flexibility and adaptability), the software defined radio design framework, GNU Radio, interfaced with Ettus products (Universal Software Radio Peripheral (USRP)) was used in this work. Three GNU Radio routines were developed to meet the design requirements of a sensor node, a primary user, and a secondary user. Two available devices from a list of Ettus products were used: 1) the USRP N210 with WBX daughterboard for sensor nodes and the secondary user (cognitive radio device) and 2) the USRP B200 for primary users. The goal is to develop an overall cognitive environment testbed and conduct an experiment to locate a secondary user by using measurements from the sensor nodes and using the primary users as points of reference. The ESRB and the CRSSB algorithms are used for position estimation.

B. RELATED WORK

Cognitive radio is the future of wireless communication; therefore, several technologies are being adopted and standardized, such as the Institute of Electrical and Electronics Engineers (IEEE) standards, 802.22 [10], [6], [11] and the 802.11af [11].

A software defined radio design approach helps promote the development of wireless communication systems based on cognitive radio features because of the capability of software defined radios to dynamically change their features and to reconfigure themselves to accommodate network requirements [12]. Consequently, a large number of research projects are being conducted to test the feasibility of cognitive radios and their ability to benignly share the spectrum with licensed users using software defined radio tools [13], [14], [15]. In this thesis, we use the Ettus USRP devices to implement a testbed of a cognitive radio system.

Source localization for cognitive radio using wireless sensor nodes and cooperative spectrum sensing algorithms remains an active area of research because current localization schemes seem to be inefficient when dealing with this type of devices. Thus, multiple solutions based on the previously mentioned approaches are proposed, such as the semi range-based location scheme, the cooperative received signal strength localization scheme and the extended semi range-based location scheme [1], [2] [9]. In this work, we adopt the ESRB and the CRSSB localization schemes to estimate the position of a cognitive radio device and to demonstrate the scheme ability to such devices.

C. THESIS OUTLINE

A background on cognitive radio characteristics and applications is provided in Chapter II, along with an overview of the software defined radio design approach and source localization schemes. In Chapter III, the conceptual diagram of the overall proposed scenario to test the ESRB and the CRSSB localization schemes is provided. The testbed scenario used to implement the ESRB and CRSSB localization scheme, along with test results, are presented in Chapter IV. A summary of the achieved work, the significant results accomplished in this work and perspectives for future work are included in Chapter V. The GNU Radio code used to perform the overall testbed development and testing is provided in the appendix.

II. BACKGROUND

In Chapter I, the cognitive radio was mentioned as a solution for the spectrum scarcity problem; however, this solution brings new challenges, especially in a source localization process. An overview of cognitive radio and source localization using a wireless radio frequency sensor network is provided in Sections A and C of this chapter, respectively. A discussion of software defined radio and an examination of its characteristics and benefits is explained in Section B.

A. COGNITIVE RADIO

In [16], the Federal Communications Commission (FCC) defines cognitive radio as

> A radio or system that senses its operational electromagnetic environment and can dynamically and autonomously adjust its radio operating parameters to modify system operation, such as maximize throughput, mitigate interference, facilitate interoperability, access secondary markets.

The FCC also dictated specific terminology for the cognitive environment in which:

- A primary user is defined as the licensed user of a specific spectrum band in a specific area; it has the highest priority and privilege of access in that band [3].
- A secondary user is defined as an unlicensed user that can opportunistically access the frequency spectrum without causing any interference to a primary user [3].
- Black spaces are bands of frequency that are occupied by a high-power signal from time-to-time; it is necessary for the secondary user to avoid using black spaces at that specific time [3].
- Grey spaces are channels occupied by a low power signal. The secondary user can consider those spaces for use in extreme needs [3].
- White spaces or spectrum holes are the opportunities that a secondary user is mainly looking for because they are signal-free except for environmental noise [3].

1. Cognitive Cycle

For a secondary user to be able to opportunistically use the white space, it must have the cognitive radio capabilities as outlined in the FCC description [16]. The cognitive radio architecture is based on the cognitive cycle. It is composed of four major interconnected functions, spectrum sensing, spectrum management, spectrum mobility, and spectrum sharing, as shown in Figure 1.

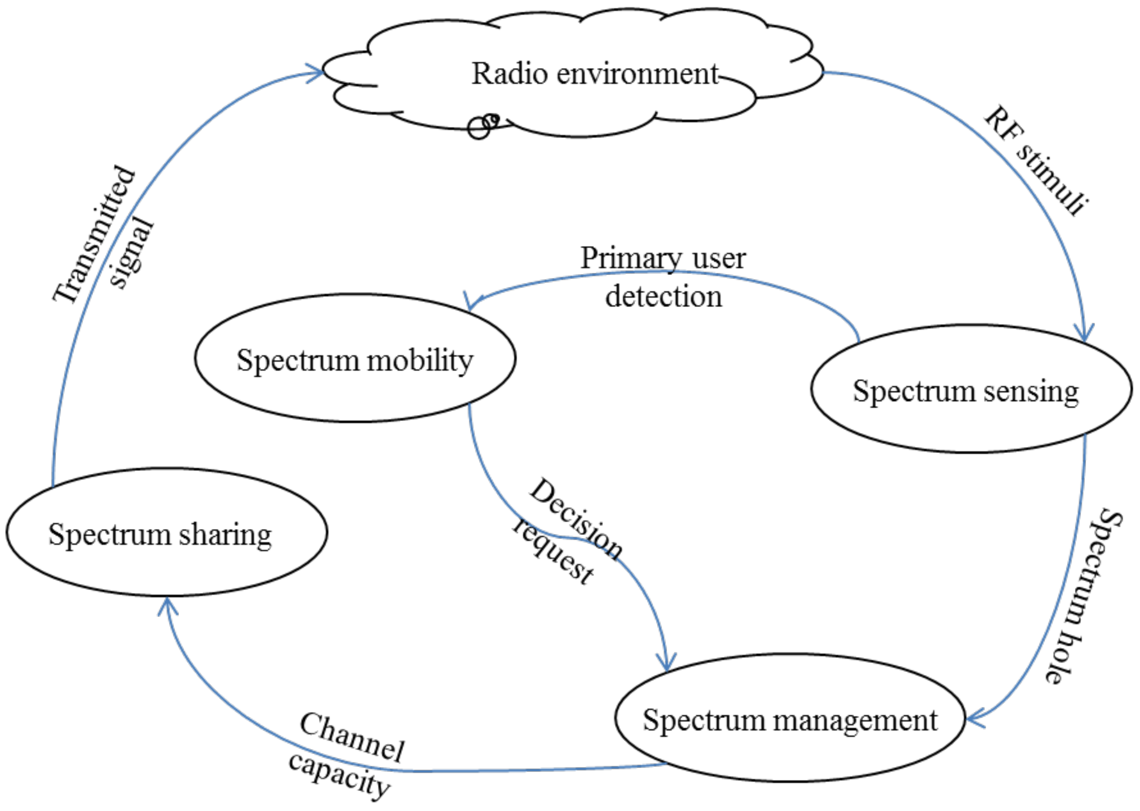

Figure 1. Cognitive cycle (from [17]).

Spectrum sensing is defined as the process that permits the cognitive radio to detect primary users, to create a picture of the spectrum occupancy and find white space that can be shared without any harmful interference between the primary and the secondary users. This is the most important process required in the cognitive radio design [17]. The next subsection is dedicated to the description of the spectrum sensing process.

Spectrum management is the task of analyzing the results of the spectrum sensing functions and deciding the best available white space that satisfies the communication quality-of-service (QoS) requirements [17]. Spectrum mobility is responsible for exchanging the secondary user's operating frequency when it is necessary to avoid interference between primary users and the secondary user [17]. Spectrum sharing is responsible for managing the use of the spectrum and guaranteeing that it is shared among the users (primary and secondary users) without any degradation on the QoS. It is the most challenging task in the cognitive radio design [17].

2. Spectrum Sensing

Observing the radio environment over a long period of time shows that its behavior is not static over time but may change at any time. In order to keep the spectrum sharing benign, secondary users must be able to back off from operating in a given frequency band whenever the primary user needs to utilize that band; therefore, the frequency band-of-interest should be periodically sensed before any access by a secondary user [14], [7], [14].

Spectrum sensing is defined in [18] as "the art of performing measurements on a part of the spectrum and forming a decision related to spectrum usage based upon the measured data." Typically, spectrum sensing provides knowledge of instantaneous occupancy of the frequency band-of-interest. This requires examining a narrow sub-band (or channel) over a short period of time in order to be able to identify whether or not a primary user is occupying this sub-band [2], [18], [19].

The following sub-sections highlight three of the most common spectrum sensing methods: 1) energy detection-based methods, 2) cyclostationary-based methods, and 3) matched filter-based-methods.

a. Energy Detection-Based Method

The energy detection spectrum sensing method is the most widely used method because of its simplicity and low computational cost [3]. Detection is based on calculating the average energy of a received signal at a particular channel over a short period of time and then comparing it to a threshold [3], [20]; hence, no prior knowledge of the signal features is required, only the noise level in the spectrum band-of-interest is needed to set up the detection threshold β to be able to determine one of the two hypotheses (H_0 or H_1):

$$P_x(t) = \begin{cases} P_n(t), & H_0 \\ hP_s(t) + P_n(t), & H_1 \end{cases}, \qquad 0 < t \leq T \tag{1}$$

where $P_x(t)$ is the power of the received signal, $P_s(t)$ is the transmitted signal power from primary user, $P_n(t)$ is the noise level in the surrounding environment, h corresponds to the channel attenuation, t is time, and T is the time period [11], [19].

In the case of hypothesis H_0, a free or unoccupied channel is detected; thus, a secondary user can opportunistically use it. In the case of hypothesis H_1, a busy or occupied channel is identified, and cannot be used by secondary users [4], [5], [19].

b. Cyclostationary-Based Method

Since any modulated signal presents a periodicity in its behavior, the cyclostationary-based spectrum sensing method offers an alternative to the energy detection based method by taking advantage of the signal statistical properties [2], [8], [20]. The detection process is realized by retrieving the cyclostationarity property of the received signal which corresponds to the unique cycle frequency, taken from the spectral correlation function given by [3]

$$S(f, \varsigma) = \int_{-\infty}^{\infty} R_x^{\varsigma}(\tau) e^{-j2\pi f \tau} d\tau \tag{2}$$

where $R_x^{\varsigma}(\tau)$ is the cyclic autocorrelation function determined by [20]

$$R_x^{\varsigma}(\tau) = E\left\{x(t+\tau)x^*(t-\tau)e^{-j2\pi\varsigma t}\right\} \tag{3}$$

$x(t)$ is the detected signal and ς is the cyclic frequency [20].

This method has more advantages than the previous method. With this technique, it is possible to differentiate among detected users (primary or secondary), and the detection of low signal-to-noise ratio (SNR) signals is feasible [3]. This approach requires *a priori* knowledge of the cyclostationary properties of the transmitted signal [3], [8].

c. Matched Filter-Based Method

The matched filter spectrum sensing technique is the optimal detection method of all the previously mentioned methods for three reasons: 1) it has the shortest processing time, 2) it achieves the lowest probability of false alarms, and 3) it makes detection possible even for low SNR signals [3], [20].

To accomplish detection, the received signal is cross-correlated with a locally generated signal similar to the transmitted one (having the same features) [3], [20]. This detection technique requires a complete knowledge of the transmitted signal, which is a drawback given that some information may be unavailable in advance [3], [20]. Additionally, the hardware implementation of this technique is very complex, especially in the case of the detection of multiple signals. The receiver's design in this case requires the use of a separate matched filter for each channel of interest [3].

3. Cooperative Spectrum Sensing

The effectiveness of spectrum sensing methods for a single sensor node is limited by the fact that a single sensor node can misidentify the presence of a primary user if the transmitted signal experiences any type of multipath fading or non-line-of-sight conditions [20]. To overcome this problem and to be able to obtain an effective global result, a cooperative spectrum sensing solution is introduced in [20]. In this approach, many sensors are dispersed to cover an area of interest and configured to share spectrum information with each other through a single decision station in which a global decision is processed [2], [20].

Three essential steps define the cooperative spectrum sensing technique [21]. First, a sensor node carries out local sensing and checks whether the sensed channel is occupied. Second, the individual sensor node decisions are sent to the decision maker node where they are collected and further processed to form a global decision on the occupancy of the sensed channel based on a predefined decision rule. For example, the logical OR rule may be used when a channel is declared busy if only one individual decision declares it so [21].

4. Application of Cognitive Radio: IEEE 802.22 Standard

Cognitive radio represents the next generation technology in wireless communications. It is a promising technique for several markets, such as public safety and military communications. The most relevant application is the implementation of an operating cognitive radio network on top of a television broadcast network. In early 2002, the IEEE 802.22 working group presented the wireless regional area network standard, which provides guidelines on using cognitive radio networks to supply broadband wireless last mile access in rural areas [20], [21].

Fundamentally, a deployed cognitive radio should not cause any interference to the existing television network (primary user); hence, those users are required to sense the spectrum before accessing channel in order to prevent collisions with the primary user [3], [6], [19], [20].

a. Wireless Regional Area Network Deployment Scenario and Cognitive Radio Architecture

A deployment scenario for wireless regional area networks is shown in Figure 2. The IEEE 802.22 standard proposes a centralized topology for the wireless regional area network (a point-to-multipoint architecture), which means that a single base station is able to manage every single station (consumer premise equipment) within its area of coverage or cell [6], [9], [22]. The base station is capable of controlling communication and media access of up to 255 consumer premise equipment terminals.

The standard proposes a multi-layer based architecture for the operating cognitive radios in the wireless regional area network, as shown in Figure 3 [6], [10], [22]. The physical layer provides the necessary functionality to support cognitive ability, such as spectrum sensing and data communication functions [6], [10].

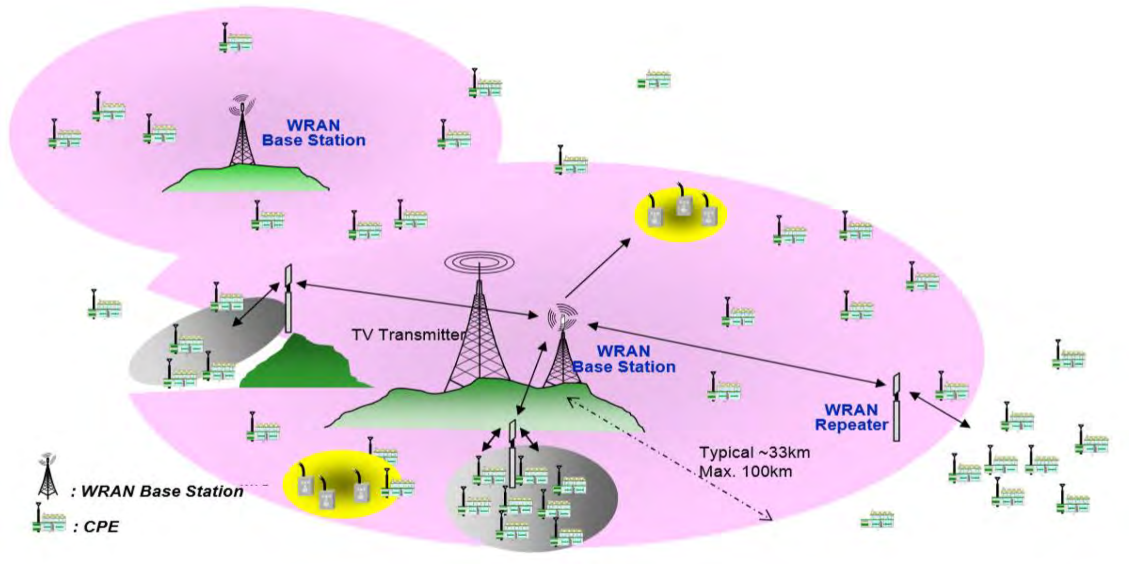

Figure 2. Deployment scenario of a wireless regional area network (WRAN) over TV network (from [22]).

Second, the medium access control (MAC) layer coordinates access to the media and synchronization between cells by managing the spectrum access that is promoted by using a superframe configuration. A superframe is composed of 16 MAC frames of ten milliseconds each, which make one superframe's duration equal to 160 milliseconds [2], [6], [10]. Finally, the higher layers (e.g., IP and ATM) are responsible for maintaining a good communication QoS [6], [10], [22].

b. Spectrum Sensing in the IEEE 802.22 Standard

The IEEE 802.22 standard dictates that cognitive radio network elements should be aware of the spectrum occupancy instantaneously. This functionality is performed using 1) the predefined television channel usage database and 2) spectrum sensing [6] [10], [22]. The cooperative spectrum sensing technique is the method suggested by the standard. The central base station is deployed as the decision-maker station, which may instruct each sensor node to carry out spectrum sensing in order to identify the occupancy of a channel of interest [20], [21].

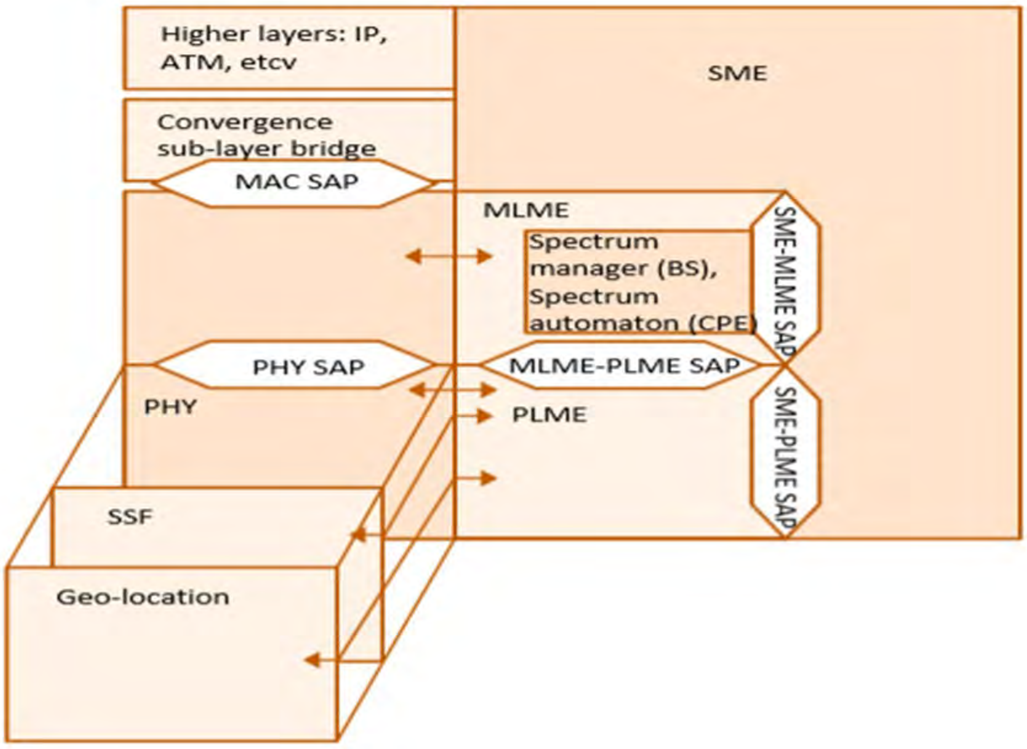
Higher layers: IP, ATM, etcv
SME
Convergence sub-layer bridge
MAC SAP
MLME
Spectrum manager (BS), Spectrum automaton (CPE)
SME-MLME SAP
PHY SAP
MLME-PLME SAP
PHY
PLME
SME-PLME SAP
SSF
Geo-location

Figure 3. Reference architecture for cognitive radio operating in IEEE 802.22 standard (from [10], [22]).

The sensing process is accomplished in two steps: coarse and the fine sensing. Coarse sensing is performed quickly (less than 1 ms) so that a general idea of the spectrum occupancy is obtained; usually, an energy-detection-based technique is used in this step. Based on the generated results, and to have a more precise measurement, the base station (decision maker) may command a sensor to execute fine sensing in a specific channel. Fine sensing is usually based on more sophisticated techniques than energy-detection-based methods (cyclostationary or matched filter based techniques) [20], [21].

B. SOFTWARE DEFINED RADIO

The increase in the pace of development of wireless communication devices has led to a variety of protocols and standards [13]. To be able to communicate with other devices operating with different network protocols, an up-to-date communication system should be able 1) to interface with any other system in the market, 2) to easily respond to upgrades of eventual innovation, and 3) to support integrated services [13]. In order for these devices to be able to set up a reliable communication with an acceptable QoS, they have to be capable of changing their features dynamically and adapting themselves to the required communication characteristics. Software Defined Radio (SDR) architecture is a satisfactory solution for the previously mentioned needs since the radio is capable of reconfiguring itself and altering its features to accommodate the network requirements [13].

1. Software Defined Radio (SDR)

In 1991, Mitola presented software defined radios that had the capability to be dynamically reprogrammed and reconfigured [13]. Later on, the Software Defined Radio Forum characterized the ultimate software radio (USR) as a radio with the ability to be fully programmable through control information and to be capable of operating over a wide frequency band [13]. A more realistic definition for software defined radios is stated as

> a software defined radio is a radio exhibiting some control on the radio frequency hardware by reprogramming some of its features, such as the modulation scheme, encryption, and error correction process. As a result, the same hardware can be used to accomplish different tasks at different times [13].

2. Software Defined Radio Model

A practical model for a software defined radio is shown in Figure 4. Its main components are 1) a flexible radio frequency hardware, 2) an analog-to-digital converter (ADC) and digital-to-analog converter (DAC), 3) a channelization and sampling rate converter, and 4) a processor (hardware and software). The use of a smart antenna permits the radio to minimize the noise and multipath fading effects on the received signal [13]. The main purpose of the flexible radio frequency hardware is to convert the received signal to an intermediate frequency in the receiver and to translate an intermediate frequency signal to the desired frequency in the transmitter [13].

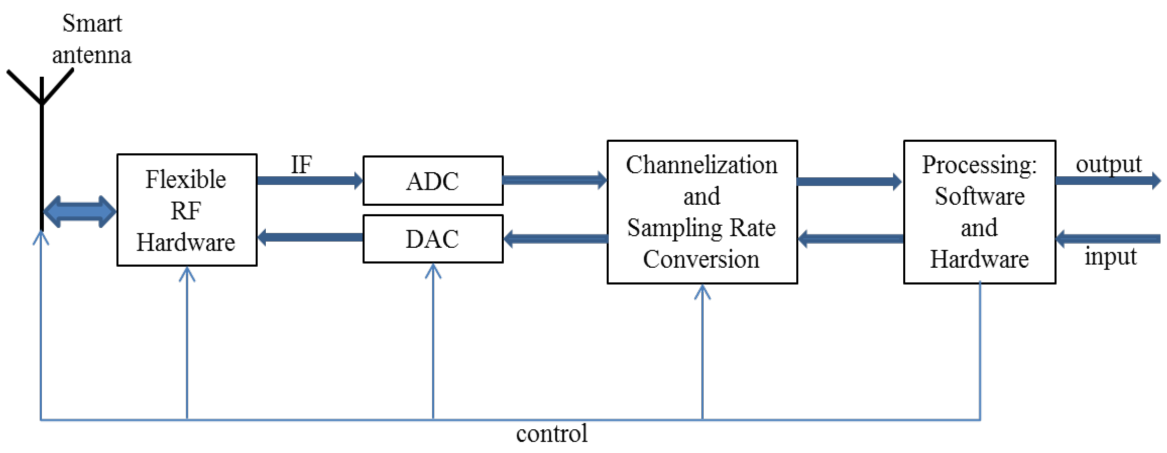
Smart antenna
Flexible RF Hardware
IF
ADC
DAC
Channelization and Sampling Rate Conversion
Processing: Software and Hardware
output
input
control

Figure 4. Software defined radio typical model (from [13]).

The analog-to-digital converters and digital-to-analog converters permit the conversion of the analog intermediate frequency signal to a digital signal and the processed digital data to an analog intermediate frequency signal, respectively. For most software defined radios operating as receivers, the conversion of the analog signal to the digital domain is done as quickly as possible to allow the maximum number of the signal processing tasks in the digital domain since digital algorithm implementations are easier than analog tasks. In case of the transmitter, most of the signal-processing tasks are carried out in the digital domain before conversion to the analog domain and transmission [13].

The channelization and sampling rate conversion block allows interfacing between the analog-to-digital converter and the processing hardware and adapts the output sampling rate of the analog-to-digital converter to the rate supported by the processing hardware (e.g., field programmable gate array) and vice versa [13]. The processing function is meant to accomplish all the digital signal processing functionalities (e.g., modulation and demodulation) using either software (e.g., GNU Radio, and Simulink) or reprogrammable hardware, such as field programmable gate arrays and application specific integrated circuits [13].

3. Benefits

Software defined radios allow service providers to easily and quickly upgrade their infrastructure to meet the requirement of integration with other networks. This can be done by taking advantage of the flexible software defined radio architecture, which allows the radio to alter its features and to meet the desired communication QoS. Additionally, software defined radios have the capability to operate in accordance with multiple standards and protocols in different regions, which defines its global mobility feature [13].

A software defined radio device is a great tool for research and development (R&D) in networking and communications fields because of its reconfigurability feature; the device may be reconfigured many times in a testbed scenario. Also, software defined radios are compact and power efficient since the same piece of hardware can be reused to perform different tasks and interfaces [13].

A large variety of software defined radio products are commercially available today. The most common products for R&D use are from the Ettus Research (USRPs) and Epiq Solutions, which are fairly inexpensive low power reconfigurable radio systems with high capability and wide frequency range. Many venders are marketing their software defined radio products for safety and military use, such as the R&S M3TR from Rohde & Schwarz and the Harris XG26P from Harris Corp.

C. LOCALIZATION USING WIRELESS RADIO FREQUENCY SENSORS NETWORK

Source localization is a very important task, especially in the case of security and military applications. Various localization techniques that permit a wireless system to locate itself or other operating wireless devices in the same neighborhood can be found in the literature [1], [23], [24]. Those schemes can be categorized as range-free and range-based localization techniques [1], [24].

Range-based localization schemes accomplish position estimation in two phases [25]. First is the ranging phase, in which the algorithms try to estimate the range between the receiver and the transmitter using one of the common metrics (e.g., time-of-arrival, time-difference-of-arrival, and received-signal-strength). Second is the localization phase, in which the position of a transmitter is estimated by intersecting three or more estimated ranges from different sensor nodes with known positions [25]. Range-free localization schemes permit estimation of the position of a radio device using a wireless sensor network; thus, multiple sensors with known positions are dispersed in the area-of-interest and configured to cooperate [26].

These two schemes are not able to provide good position estimations in the case of cognitive radio localization [27]. This is because both techniques lack the capability to change their features as the cognitive radio changes. Consequently, any scheme meant to locate a cognitive radio and accurately estimate its position must support some level of adaptation and be able to account for the capability of the target radio to hop from one frequency to another over time [27]. Semi-range based localization is a feasible solution for this problem.

1. Semi-Range-Based Localization Scheme

This scheme was proposed to estimate the position of a primary user in a cognitive radio environment [24]. The secondary users in this case form a wireless sensor network to perform cooperative spectrum sensing. The results are then used to draw a map of the spectrum occupancy, and the map is used to estimate the location of the desired primary user [24].

Given that the position of each sensor node is known in advance, the scheme relies on exploiting the relationship between the probability of detection and the distance of the secondary user to the primary user [24]. This technique accomplishes location estimations by taking advantage of both range-based and range-free localization estimation methods. The processing is performed in two steps. First, the probability of detection for a primary user is estimated using the binary decision of local spectrum sensing reported by each sensor node (secondary user in this case). Second, the position of the desired primary user is estimated using the probability of detection and the received-signal level, similar to the way estimation is carried out by a range-based scheme [24].

To be highly accurate, the semi-range localization scheme requires *a priori* knowledge of the transmitted power by the primary user, which is a major drawback of this technique because it violates the fundamentals of cognitive radio environment; no cooperation is allowed between primary user and secondary user [2]. A solution to this problem was proposed in [23] as a practical semi range-based localization method. This algorithm reduces the need for *a priori* knowledge of the transmitted signal power by estimating it during the localization process using the non-linear-least-square method; however, neither technique provides an accurate position estimate, especially in the case of locating a secondary user in a cognitive radio environment [2].

2. Extended Semi-Range-Based Localization Scheme

In [1], an extended semi-range-based (ESRB) localization scheme was proposed to accurately estimate the position of cognitive radio using wireless sensor network. The conceptual diagram of the ESRB is shown in Figure 5. The algorithm relies on four primary aspects: 1) cooperative spectrum sensing, 2) spectral environment mapping, 3) localization through the iterative nonlinear least-squared method, and 4) position refinement [1], [2]. Overviews of each aspect of the functionalities are provided in the following subsections.

a. Spectrum Sensing

This task takes place at each sensor node of the wireless sensor network in order to determine if channels are occupied over a period of time [2]; therefore, the sensor node performs an energy detection process at each channel. The decision data is recorded into a spectral scanning report in which occupied channels are identified using a binary ‘1’, and unoccupied channels are identified using a binary ‘0.’ After the overall spectrum of interest is scanned, the scan report is transferred to the decision maker for further processing [2].

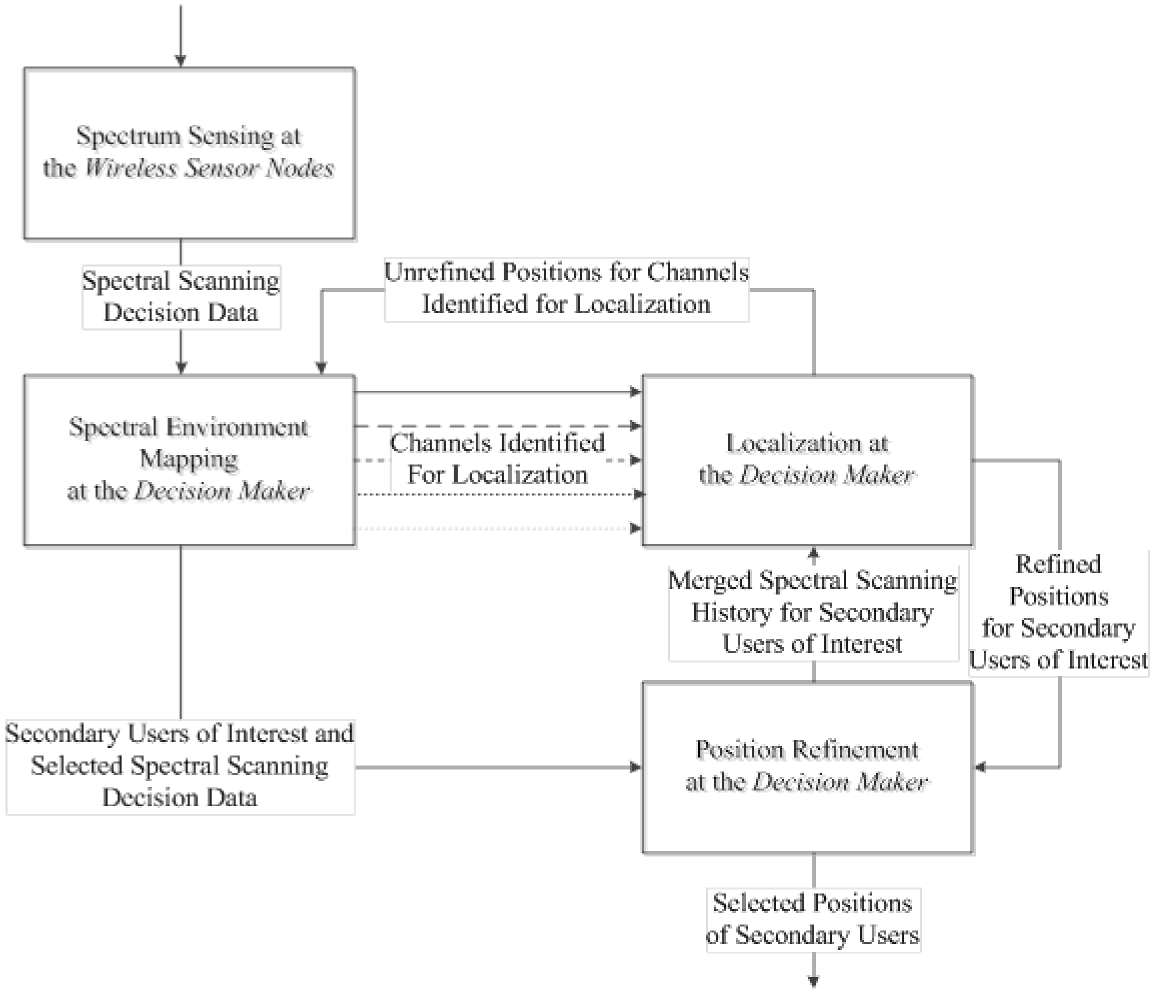
Spectrum Sensing at the *Wireless Sensor Nodes*
Spectral Scanning Decision Data
Unrefined Positions for Channels Identified for Localization
Spectral Environment Mapping at the *Decision Maker*
Channels Identified For Localization
Localization at the *Decision Maker*
Merged Spectral Scanning History for Secondary Users of Interest
Refined Positions for Secondary Users of Interest
Secondary Users of Interest and Selected Spectral Scanning Decision Data
Position Refinement at the *Decision Maker*
Selected Positions of Secondary Users

Figure 5. Conceptual diagram of the proposed extended-semi-range-based (ESRB) localization scheme for cognitive radio positioning (from [2]).

b. Spectral Environment Mapping

This process is carried out at the decision maker and is performed by interpreting the collected scan reports from each sensor node [2]. The main goal of this task is to differentiate between occupied and unoccupied channels by drawing the spectral environment map; thus, a cooperative spectral sensing process is executed. In order to optimize the detection algorithm efficiency, the majority decision rule is the adopted approach, in which a channel is declared as occupied if the number of sensors indicating that it is a busy channel is more than half of the total number of sensor nodes. Only identified busy channels with their corresponding signal level are transferred to the next processing level [2].

c. Localization

The main purpose of this task is to identify whether the present user is a primary or a secondary user. For each occupied channel, an estimation of the present user position is calculated and compared to previously known primary users' positions (available in a geo-location database) [2]. Any estimated position that matches within an acceptable error (predefined level of tolerance) with any available position in the geo-localization data base is discarded. If the position estimate does not match with any primary user position, it is considered a potential secondary user or user-of-interest, and its position estimate is stored to form the history and is fed to the position refinement process [2].

d. Position Refinement

The intention behind this process is to evaluate the results of the previous process to provide accurate positions for the secondary users [2]. The position refinement process manages the history of the discovered user-of-interest. For all received data, the process tries to determine if any of the new position estimates match with old positions within a radius of tolerance. Matched positions are merged together, and positions that have been recorded multiple times are declared to be a secondary user. If no match is found, the newly discovered position is recorded as a new secondary user, and the estimated position is entered in the history record [2].

3. Cooperative-Received-Signal-Strength-Based Localization Schemes

In the cooperative-received-signal-strength-based (CRSSB) localization schemes, the distance between the transmitter and the receiver is estimated based the calculated squared-magnitude of the signal and the channel propagation attenuation model [9]. To be accurate on distance estimation, any localization scheme based on received signal strength requires an accurate channel propagation model. Classical received-signal-strength-based localization schemes require *a priori* knowledge of the effective isotropic radiated power of the transmitter to obtain an acceptable location estimation. This is a drawback of the technique, especially when the effective isotropic radiated power of the transmitter of interest is unknown; however, this kind of scheme is considered a low cost localization technique because it is relatively easy to implement.

In [9], an algorithm using the received-signal-strength metric without any knowledge of the effective isotropic radiated power of transmitter in advance is proposed in order to optimize the effectiveness of this scheme in cognitive environment. First, all sensor nodes apply a fast spectrum sensing (for a short period-of-time) to obtain an idea of the occupancy of the spectrum-of-interest and report the calculated energy at the channel to the decision maker. Based on those energies, the decision maker decides which sensor nodes need to apply an additional fine spectrum sensing (nodes with the highest energy are chosen). Second, the chosen sensor nodes carry out a fine spectrum sensing to determine a more accurate energy estimation of the signal occupying the channel and report the estimated energy to the decision maker. Third, the decision maker uses the received energy estimates and the positions of the sensor nodes to estimate the transmitter positions using a received-signal-strength technique. Finally, the estimated positions are compared to the primary user positions. If a match is found within an acceptable level of tolerance, the position estimate is discarded. If no matched is found, the estimate becomes the position of a potential secondary user [9].

In this chapter, an overview of cognitive radio characteristics and applications was presented to illustrate how this concept can be used to overcome the problem of spectrum scarcity, and an outline of software-defined radio characteristics and benefits was provided. Multiple source localization schemes were introduced, along with an explanation of the ESRB and CRSSB localization schemes for cognitive radio. In Chapter III, a conceptual design of a cognitive radio environment is proposed to implement and test the feasibility of the ESRB and the CRSSB localization schemes.

III. COGNITIVE RADIO ENVIRONMENT CONCEPTUAL DESIGN

The main advantage of a cognitive radio is its ability to modify its attributes over time (e.g., frequency and modulation) in order to adapt to the surrounding environment and avoid interference with primary users [1], [6], [11]; however, source localization is very challenging when considering this type of device for two reasons. First, any source localization scheme must be able to adapt along with the cognitive radio as it changes. Second, it requires keeping track of the radio's frequency occupancy to distinguish between primary and secondary users of the frequency spectrum [2], [24]. The ESRB and CRSSB localization schemes were proposed in [1] and [9], respectively, to overcome the challenge of identifying and tracking the position of a cognitive radio over time.

An overview of the proposed software-defined radio testbed and its schematic diagram are given in Section A of this chapter. An outline of the scenario design along with a detailed explanation of the design principals of each element of the cognitive radio system are provided in Section B of this chapter. Finally, the decision-maker design is presented in Section C.

A. PROPOSED SCHEME

To test the feasibility of the ESRB localization scheme, a scenario was introduced in [1] that demonstrated how a wireless sensor network can be used to locate and track a cognitive radio over time. The scheme's underlying principle is the monitoring of the environment's temporal parameters (i.e., position and frequency occupancy) in a collaborative manner to determine the cognitive radio's position [1]. To accomplish this, the scheme relies on multiple sensor nodes to create a wireless radio frequency sensor network. The collected measurements from each sensor are used in a collaborative manner to obtain spectrum sensing results. These results are in turn used to estimate the position of the emitter-of-interest (cognitive radio) [1], [2].

The developed software-defined radio-based cognitive radio system used to implement the testing scenario is explained in this section. The schematic diagram of the proposed system is shown in Figure 6. The proposed hardware testbed system consists of four major parts: the primary users who have the right to access a frequency band-of-interest, a secondary user who can opportunistically access the same frequency band when the primary users are idle, a sensor network consisting of multiple radio frequency sensor nodes that continuously measure the signal strengths of both the primary and secondary users, and a location estimation scheme to determine the position of the secondary user.

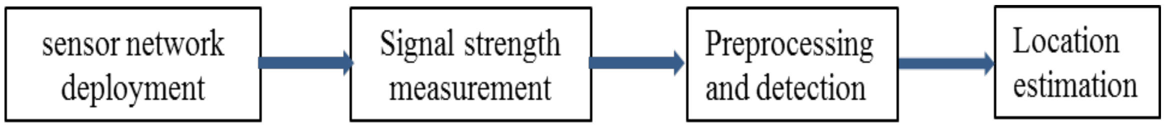
sensor network deployment
Signal strength measurement
Preprocessing and detection
Location estimation

Figure 6. Proposed scheme for location estimation of a CR in a dynamic frequency environment.

First, a wireless sensor network is deployed in an area-of-interest in which primary users and a single secondary user are sharing the same frequency band. Second, each sensor node performs spectrum sensing in the band of interest to determine whether a user is present by comparing the measured signal to a preselected threshold. Third, a preprocessing and detection process is carried out in order to differentiate between the primary user and the secondary user. Finally, the localization estimation process is accomplished using the output of the previous process and the ESRB or the CRSSB localization algorithms. Each function of the proposed scheme is explained in detail in the following sections.

B. SCENARIO DESIGN

The goal of the proposed scheme is to estimate the location of the secondary user in a cognitive radio environment and to track the secondary user's frequency occupancy over time using a collaborative spectrum sensing approach [1], [2]. The scenario is designed to test the performance of the ESRB and the CRSSB schemes using a wireless sensor network as illustrated in Figure 7. The testing scenario consists of multiple sensor nodes that are randomly distributed in an area-of-interest (the secondary user environment) in which several primary users are present and a single secondary user is deployed. Additionally, a decision maker is located within the sensor nodes in order to process the collected measurement and return the estimated position of the secondary user. All of the primary users and sensor nodes' positions are assumed to be known in advance and stored in a geo-localization database [2].

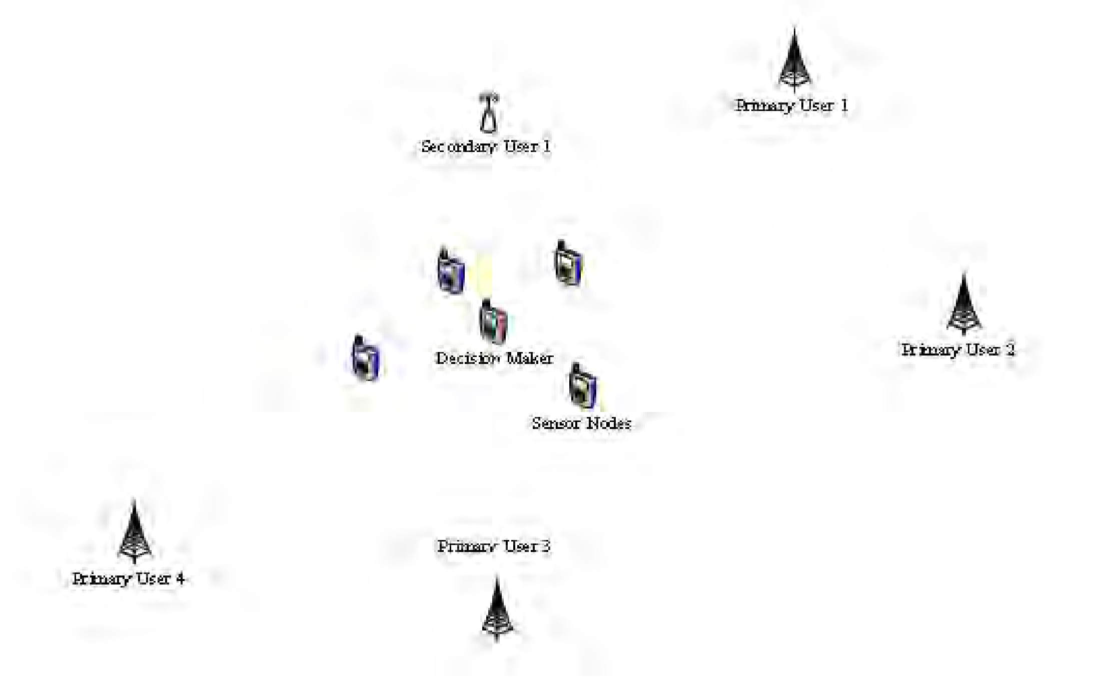

Figure 7. Geolocation scenario for cognitive radio using a wireless radio frequency sensor network (from [2]).

This scenario is based on IEEE 802.22 standard. An overview of the design of each component is provided in the following subsections.

1. Wireless Sensor Network

Multiple sensor nodes are deployed in the area-of-interest and connected to the decision maker, which together form a wireless radio frequency sensor network. A sensor node consists of a radio frequency sensor to measure the signal from the primary and secondary users, a transceiver to send/receive the measurements as appropriate, and a processor to undertake any local processing of the measurements. The role of each sensor node is to examine the whole frequency spectrum-of-interest and send a spectral scan report to the decision maker [1], [2]. The spectral scan report consists of a binary 0 or 1 reflecting the estimated channel energy as either vacant or occupied, respectively [2]. Multi-bit spectral scan reports utilizing multiple threshold levels are also possible. For example, a 2-bit scheme uses three threshold levels. While the complexity of implementation increases, these schemes have been shown to provide improved performance [20].

The spectral scanning process is done in three steps: tune, listen, and decide [6]. To determine the occupancy of the spectrum, an energy detection approach is adopted in this work [2] because it is the least complicated in the implementation process compared to other spectrum sensing schemes (e.g., matched filter detection and cyclostationary detection) [3], [20]. No prior knowledge (modulation scheme) of the signal is required to confirm its absence or presence [3], [20].

A complete spectrum scan consists of an examination of each frequency channel-of-interest in which the signal energy E is calculated as

$$E = 10\log_{10}\left(\frac{\sum_{i=1}^{N}(I_i^2 + Q_i^2)}{N}\right) \tag{4}$$

where I_i and Q_i are the ith symbol's in-phase and quadrature components, respectively, and N is the total number of samples.

If $E \geq \beta$ (where β is a preselected threshold value), a signal is assumed to be present (i.e., we have a busy channel) and an associated binary '1', the channel energy, and the channel number are added to the scan report [2]. If $E < \beta$, then no signal is present (i.e., we have a free channel), so a binary '0' is associated with the channel number and added to the scan report. The sensor then tunes to the next channel and repeats the same steps. After the whole spectrum-of-interest has been surveyed, a finished scan report is sent to the decision maker for further processing. This spectral scanning process is repeated indefinitely as shown in Figure 8 [2].

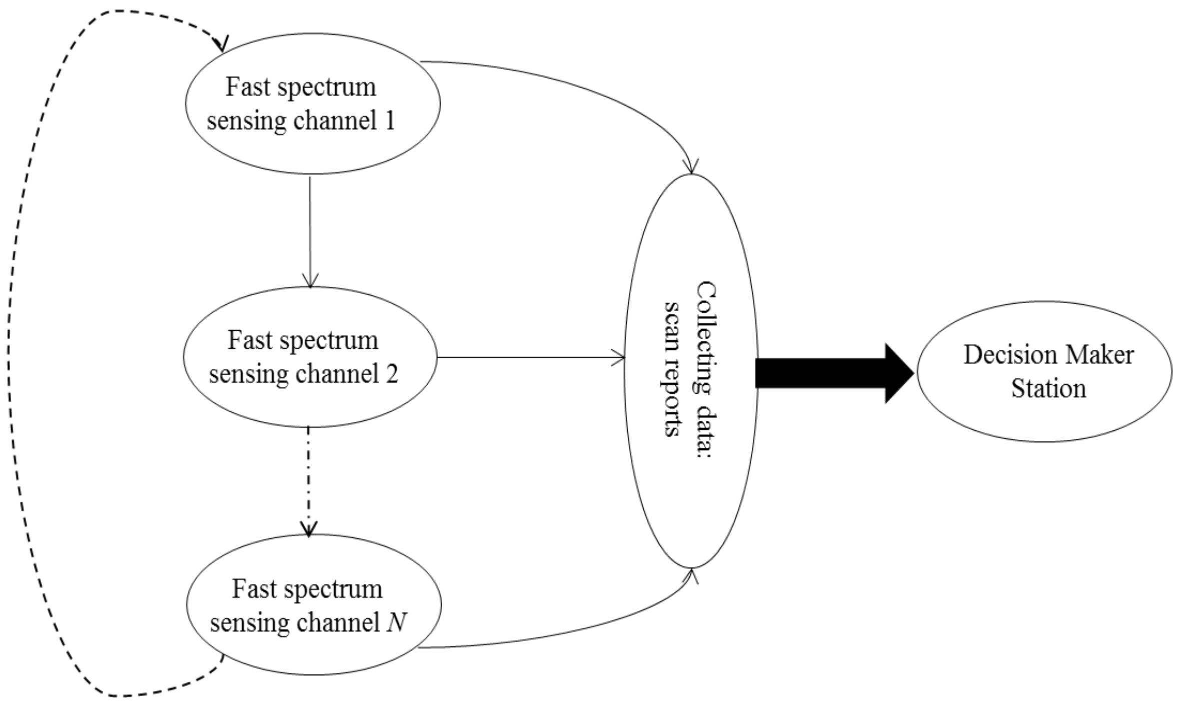
Fast spectrum sensing channel 1
Fast spectrum sensing channel 2
Fast spectrum sensing channel N
Collecting data: scan reports
Decision Maker Station

Figure 8. Sensors node state diagram

2. Primary User Network

The primary user network is composed of multiple primary users, each with an allocated fixed frequency band or channel. A frequency-division multiple accesses (FDMA) approach is adopted in the primary users' network design to ensure that adjacent channels do not overlap [2]. This design approach is also followed by the sensor nodes and is integral to the energy detection spectrum sensing approach because sensor nodes do not have the capability to differentiate among the primary user signals.

In terms of occupancy, the primary user's behavior follows a two-state Markov model as shown in Figure 9 [2], [24]. The primary user alternates between the idle and busy states (p_i and p_b are the respective transition probabilities) for different periods of time. During the idle state, no traffic is broadcast, leaving unoccupied frequency bands available (white spaces) for either a short period-of-time between the adjacent superframes or for the duration of a complete superframe. During the busy state, the primary user transmits a fixed amplitude signal in order to keep the channel occupied for a complete superframe [2], [24].

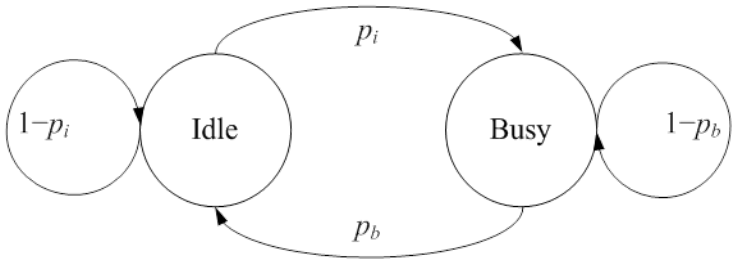

Figure 9. Two-state Markov model of primary users channel occupancy; p_i and p_b are the state transition probabilities (from [2]).

3. Secondary User

The design of the secondary user (cognitive radio) is based on the IEEE 802.22 standard [6], [10]. Most of the definitions and functions for the cognitive radio building blocks and their interconnection are provided by the standard and were taken into consideration in designing the secondary user for this scenario. The cognitive radio system state diagram consists of three major components: spectrum sensing, decision making, and data transmission as shown in Figure 10 [6], [10].

a. Spectrum Sensing

To minimize the probability of interference with primary users, the IEEE 802.22 standard dictates use of a coarse and fine spectrum sensing approach in the secondary user's design as mentioned in Chapter II [2], [7], [10]. First, coarse sensing is carried out using energy detection. Based on results of the coarse sensing and in order to have more accurate measurements of the spectrum occupancy, fine sensing may be carried out using other spectrum sensing methods (e.g., matched filter detection or cyclostationary) [2], [7], [10].

Only coarse spectrum sensing is adopted in this work since fine sensing requires the implementation of more complex algorithms to determine the primary user signal characteristics [2]. The cognitive radio carries out in-band sensing without identifying a specific modulation technique. Afterwards, the estimated signal energy is transferred to the decision making block [2].

b. Decision Making

In this step, the received energy is compared to the threshold in order to identify the presence of the primary user or white space. If a busy channel is identified, the spectrum sensing process is carried out in the adjacent channel. If a free channel is identified, the center frequency of the channel is sent to the data transmission block [2].

c. Data Transmission

The focus of this step is to generate and send data. The data generation process adopted in the scenario is a random binary packet generation process where multiple packets are placed within the length of a frame. Next, all packets are transferred to the transmission process in order to be transmitted [2].

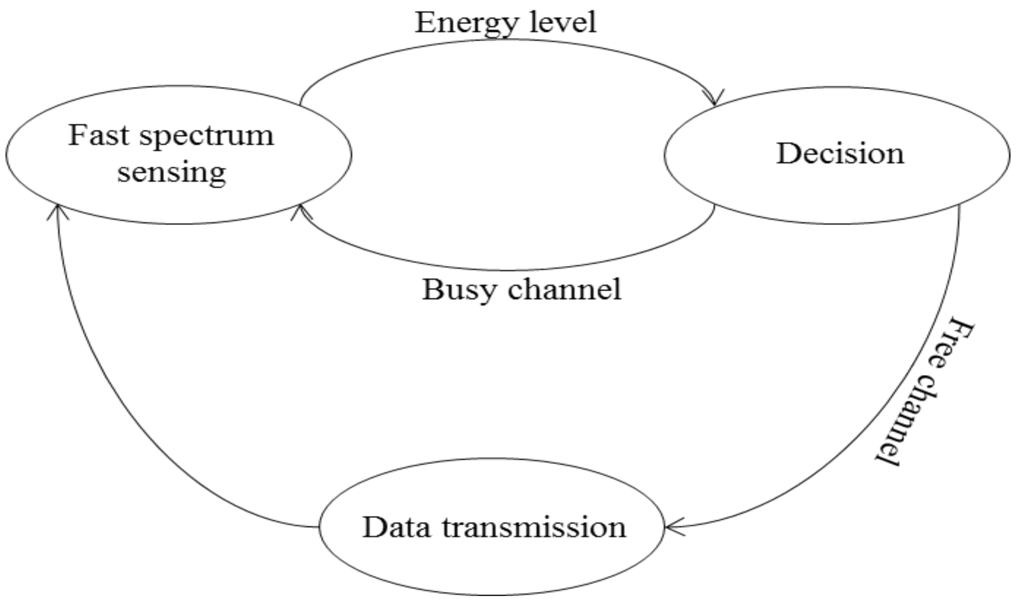

Figure 10. Secondary user state diagram

C. DECISION MAKER

The main function of the decision maker is to estimate the secondary user's position in the surrounding environment. The ESRB and the CRSSB localization scheme is adopted for this purpose, which compares the calculated position to that of the primary user's known positions. Users at unknown locations are assumed to be secondary users [1], [2], [9].

1. Case 1: Extended Semi-Range-Based Localization Scheme

After the sensor nodes conduct spectrum sensing and report their results, the decision maker develops a global spectrum occupancy map by aggregating the scan results of the entire wireless sensor network [2]. Then the decision maker identifies occupied channels over periods of time and attempts to discriminate the users within each of the occupied channels. That is, the decision maker attempts to determine which of the users is a primary user or a potential secondary user. After completing user discrimination, all potential secondary users, which are now users-of-interest, along with their recorded measurements (i.e., estimated position, estimated signal level, channel occupancy) are combined into a user-of-interest activity history [1], [2].

When the next spectral scan is received, the decision maker repeats the previous steps and compares the newly estimated position to the previously stored reference position. If no match is found, the data is discarded; however, if the new position matches within acceptable level of tolerance with the reference position, the two results are merged together to form an updated user-of-interest activity history [1], [2].

This updated history is fed back into a refinement position process where the estimated positions of all secondary users are calculated. The final step is a cross-reference of the calculated potential secondary user position with primary user geo-localization data-base to ensure that the results do not overlap with a primary user. All of the estimated positions are confirmed if they have been validated for multiple iterations [2].

2. Case 2: Cooperative-Received-Signal-Strength-Based Scheme

All sensor nodes apply fast spectrum sensing, calculate the energy at the channel, and report the calculated energy to the decision maker. Based on those energy values, the decision maker decides which sensor nodes have to apply an additional fine spectrum sensing (nodes with the highest energy are chosen). The chosen sensor nodes carry out fine spectrum sensing to determine a more accurate energy estimation of the signal occupying the channel and report the estimated energy to the decision maker. In this work, we only adopted the coarse sensing technique using the energy detection based method because implementation of fine spectrum sensing using cyclostationary or matched filter methods requires a computationally intensive algorithm [9].

The decision maker uses the received energy values and the positions of the sensor nodes to estimate the transmitter positions using a received-signal-strength technique. Finally, the estimated positions are compared to the primary user positions (from the geo-localization database). If a match is found within an acceptable level-of-tolerance, the position estimate is discarded. If no matched is found, the position estimate is considered the position of a potential secondary user [9].

In this chapter, a discussion of the proposed scheme to validate the ESRB and CRSSB localization algorithms were provided. Each component of the environment was examined, and a conceptual diagram behind each design was given. In the next chapter, the performances of the proposed schemes are demonstrated through a real world implementation. The scenario testbed design and an analysis of the physical implementation of the preceding components are provided. Intermediate test results that are used to validate the component's operating parameters are contained in Chapter IV as well.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION MODEL AND RESULTS

An explanation of the conceptual model, designed to demonstrate the performance of the ESRP localization scheme, was provided in Chapter III. All functions were explained for each of the elements: 1) sensor node, 2) primary users, 3) secondary user, and 4) decision maker. The implementation of the proposed testing scenario is presented in this chapter. An overview of the developed experimental platform is provided in Section A. The proposed testbed and implementation of each element are described in detail in Section B. The overall testing scenario, results, and discussion are presented in Section C.

A. EXPERIMENTAL PLATFORM

The platform used for the physical implementation of each element of the testing scenario is described in this section. All elements were developed through software defined radios. Part of the signal processing design was accomplished by the host machine (laptop) using GNU Radio programming software, while the other part was undertaken by the Universal Software Radio Peripheral (USRP). In the first subsection, a description of the hardware used is provided. An overview of the GNU Radio software and how it was used to design each baseband network element is given in the second subsection.

1. USRP

The Universal Software Radio Peripheral (USRP) is a hardware device developed by Ettus Research which gives engineers the capability to develop and implement flexible software defined radios rapidly and with low cost [19]. In short, a software defined radio is a radio system which performs the required baseband signal processing tasks (e.g., modulation, demodulation, filtering) in a software platform instead of using dedicated hardware integrated circuits. The remainder of the digital signal processing tasks (e.g., up- and down-sampling and digital-to-analog converter (DAC)/analog-to-digital converter (ADC)) is accomplished via reprogrammable hardware [12]. Since any software design can easily be replaced in this kind of radio system, the same hardware can be used to create many communication devices with different transmission standards, even those requiring high radio frequency performance and large bandwidth as needed in dynamic spectrum access for cognitive radios [4], [12], [14].

Ettus products are designed with a modular architecture, using a motherboard and daughterboard cards. The motherboard accomplishes some of the baseband processing on signals, such as the conversion of the signal from analog to digital [28]; however, the daughterboard permits performing analog operations on the signal (e.g., analog filtering, etc.). A large variety of daughterboards can be used with USRPs, depending upon the frequency range needed by the user. A photograph of the USRP N210 is shown in Figure 11 [28].

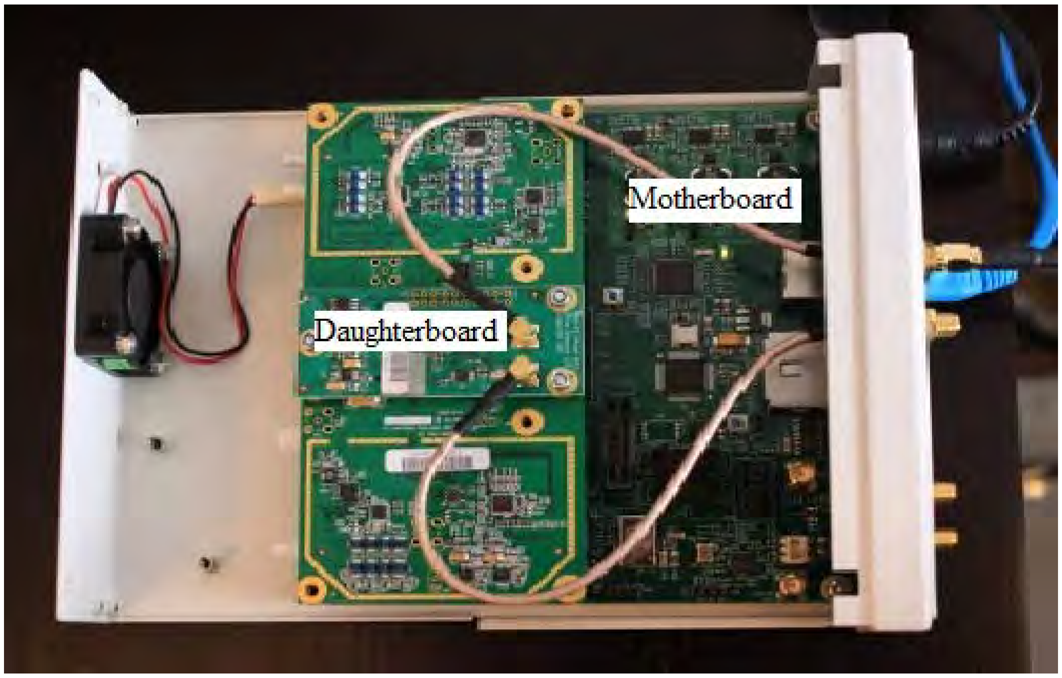

Figure 11. USRP N210 with WBX daughterboard.

Ettus Research also developed two open source applications that are useful with their products. The USRP Hardware Driver provides the USRP with the ability to communicate with several platforms (Windows, Linux, and Mac OS) [26]. The USRP Application Programming Interface can be used by multiple software frameworks (GNU Radio, Simulink). Both applications provide many useful scripts [28]. In particular, the *find_devices* script is useful because it enables a host machine to discover any connected devices and return their type and features [28].

Two devices from Ettus' list of available products are used in this work: the N210 radio with WBX daughterboard and the B200. The main features of these two devices are summarized in Table 1, where Msps is used to indicate mega-samples per second [26].

Table 1. USRP N210 and B200 features.

	N210+WBX daughterboard	B200
Interface	Gigabit Ethernet	USB 3.0/2.0
ADC sample rate	100 Msps	61.44 Msps
DAC sample rate	400 Msps	61.44 Msps
Host sample rate	25 Msps	61.44 Msps
Power output	15 dBm	>10 dBm
Received noise figure	5 dB	<8 dB
Frequency range	50 MHz-22 GHz	70 MHz-6 GHz

Both devices cover the bandwidth chosen to carry out the experimental testing in this work, the Industrial, Scientific and Medical (ISM) radio band (902-928 MHz). Each device is capable of introducing an internal noise source to the received signal. For example, the N210, which was used as a sensor node in this experiment, adds a noise figure of 5 dB.

2. GNU Radio

GNU Radio is an open-source software package developed for the implementation of signal processing and communication applications [19]. The official development project started at the Massachusetts Institute of Technology in 2000 and continues to be updated with new releases periodically [19], [29].

The software contains a large number of widely used signal processing routines denoted as blocks (e.g., filter, modulators, and demodulators) which are written in C++ [19], [29]. Additionally, it has many Python scripts that can be used to tie the blocks together to form the baseband part of any desired radio configuration. Integration between Python and C++ is controlled by a Simplified Wrapper and Interface Generator (SWIG) [19].

Given the application, a routine can be developed as a collection of signal processing blocks tied together in simulation or can be implemented with a hardware device to form a software-defined radio. In the case of a receiver, as shown in Figure 12 [30], the captured radio frequency signal is converted to an intermediate frequency by the radio frequency front end and passed through an analog-to-digital converter to be digitized. Complex samples go through a field-programmable gate array (FPGA) for digital signal processing tasks (e.g., data rate conversion) and pass to the host machine through either a Gigabit Ethernet or USB cable, where baseband processing tasks are executed. For transmission, the entire procedure is reversed. The preprocessed baseband samples are passed to the USRP from the host machine for further signal processing and digital-to-analog conversion and then converted to the desired RF frequency and transmitted [28].

B. TESTBED IMPLEMENTATION

An in-depth explanation of each network element’s design and implementation is provided in this section.

1. Primary User

As mentioned in Chapter III, the main role of the primary user element is to transmit a fixed amplitude signal for the duration of one superframe.

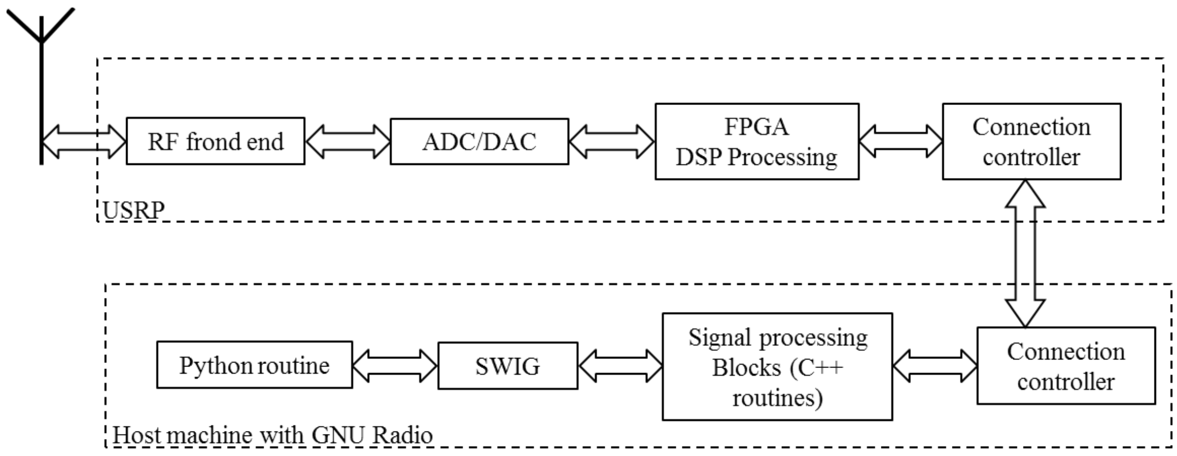

Figure 12. USRP and GNU Radio blocks and interconnections for software defined radio (from [28]).

A more detailed analysis of how the design of the primary user routine accomplishes this requirement is given in this subsection. The *banchmarck_tx.py* script from the GNU Radio example library is modified to obtain the *primary.py* python routine and satisfy the design requirements.

The purpose of the overall set up is to be able to locate a secondary user in a cognitive environment using a wireless sensor network. Given that energy detection is the only method of spectrum sensing being utilized by the wireless sensor network, the content of the primary user's data does not matter because the sensor nodes are only measuring the signal energy level. For this implementation, the data used for transmission is randomly generated by applying the python line of code:

*data = (pkt_size - 2) * chr(pktno & 0xff)*

where *chr* is a python function which returns a string of one character whose ASCII code corresponds to an integer which is generated by using a binary AND on the packet number (*pktno*) and an *0xff* hexadecimal number (matching with the decimal 255) [29]. The character is then multiplied by the packet size minus two bytes to form the packet's payload. The minus two is to account for the two-byte header.

In this work, the adopted design for data transmission is shown in Figure 13. First, data is generated as a random binary sequence and then put into 4000-byte packets using the GNU Radio function *struct.pack* (the maximum allowed packet size for this function is 4096 bytes) which adds a two bytes header. All packets are sent to the modulator block using the *send_pkt* function from the *packet_transmitter* class, which provides a variety of modulation schemes that can be used for transmission (e.g., Phase-Shift Keying (PSK) and Quadrature Amplitude Modulation (QAM)). The baseband signal is then sent to the USRP in the form of in-phase (I) and quadrature (Q) complex samples to be further processed and transmitted over the air.

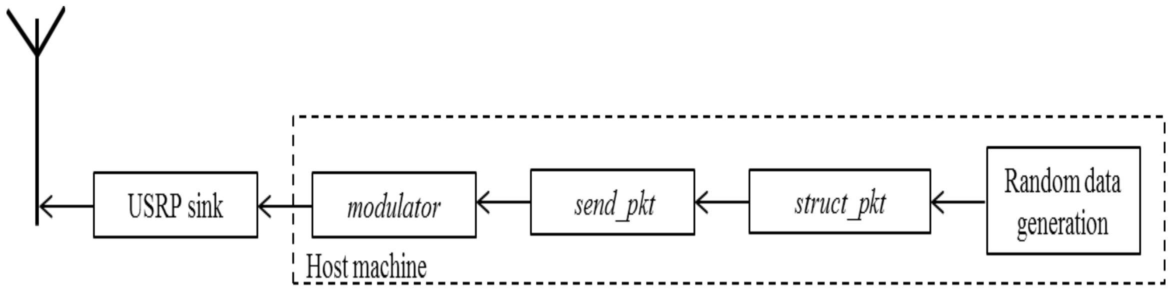

Figure 13. Transmitter flow graph

The number of transmitted packets per burst N, the packet size L, and the USRP sampling rate F_S define the superframe length, or burst duration T_{SF}, as

$$T_{SF} = \frac{8NL}{F_S}. \tag{5}$$

What matters most to the ESRP source localization algorithm is the number of scans that a sensor can achieve during one superframe. In this work, the duration of a superframe corresponds to the number of scans that a sensor can achieve during the duration of one superframe. To be efficient, the ESRP source localization algorithm requires a large number of scans per superframe. An experiment was conducted to test the relationship between the number of packets used to build one superframe and the number of scans completed. The results of this experiment are shown in Figure 14. The chosen superframe duration corresponds to the highlighted point in the curve, which corresponds to 55 scans (210 transmitted packets).

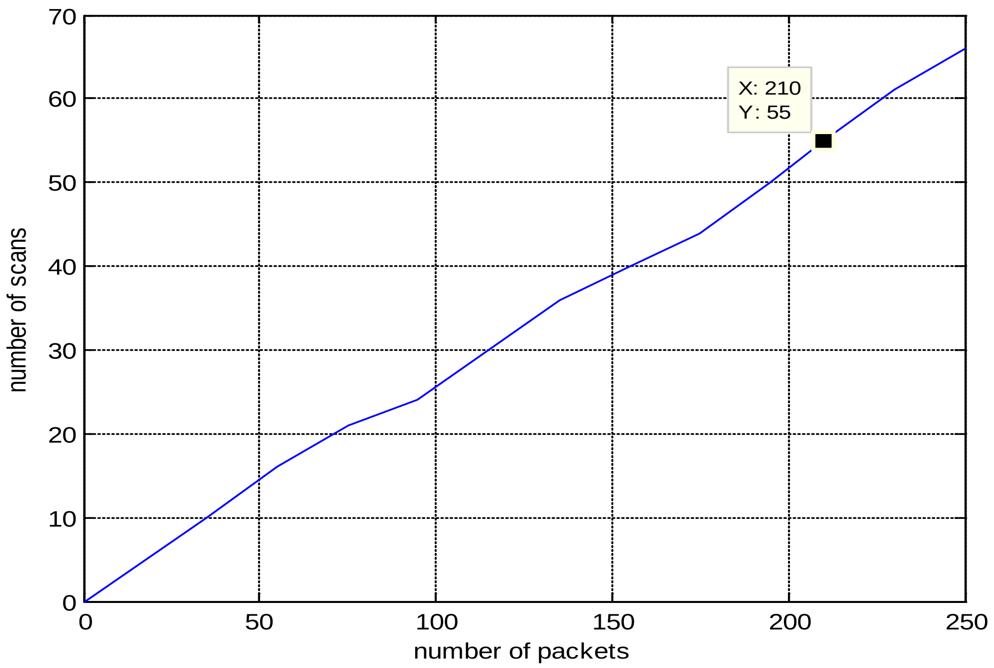

Figure 14. Relationship between number of generated packet and scan reports for one superframe duration.

Moreover, as mentioned in Chapter III, the primary user leaves portions of the frequency spectrum unoccupied (white space) at random intervals, which the secondary user can opportunistically share. To induce white space in the spectrum, a variable sleep time was set up between zero and two consecutive burst transmissions. Multiple measurements were carried out to help choose accurate values corresponding to a short quiet period between two consecutive superframes and a long quiet period equal to one or more superframe durations. The result of this is shown in Figure 15.

Experimental testing showed that, for chosen short sleep periods, the signal of two consecutive bursts appeared to be a continuous wave. This can be explained by the fact that the USRP introduces a processing delay before transmitting packets. The short quiet period is added and corresponds to point A in the plot (13 s).

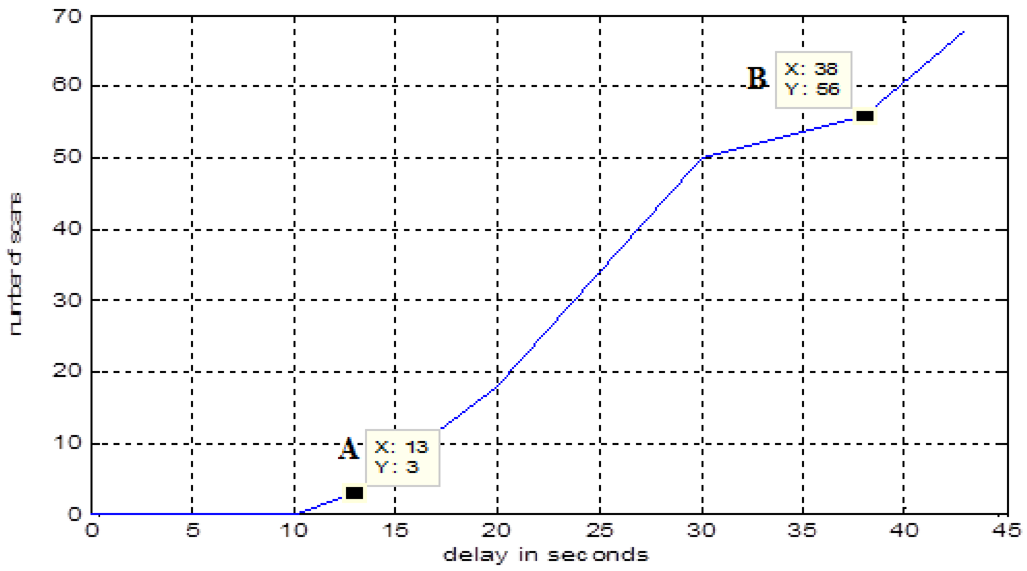

Figure 15. Relationship between time delay and number of scan reports for quiet period.

To realize variable large quiet periods that are equal to one or more superframe duration, additional testing was carried out to measure the sleeping time corresponding to 55 scans (one superframe duration), which corresponds to point B in the plot (38 s). The 38-s value is in fact a summation of the previously mentioned short quiet period (13 s) and the actual superframe duration (25 s). To have the desired random variation of large quiet periods, the measured duration of the superframe's actual value is multiplied by a random integer value between 0 and three. The above process is coded in python using these two lines of code:

k=random.randint(0,3)

*time.sleep(13+k*25).*

The *random.randint(0,3)* is a python function that returns a randomly (uniform distribution) selected element from the list (0, 1, 2, 3), and the *time.sleep* suspends the code execution for the given time delay.

2. Sensor Node

An analysis and a detailed explanation of the routine *sensor.py* running the algorithm shown in the flow graph of Figure 16 is presented in this section. The routine was developed to accomplish the requirements of a sensor node stipulated in Chapter III and is based on an available python script in the GNU Radio library entitled *usrp_spectrum_sensing.py*.

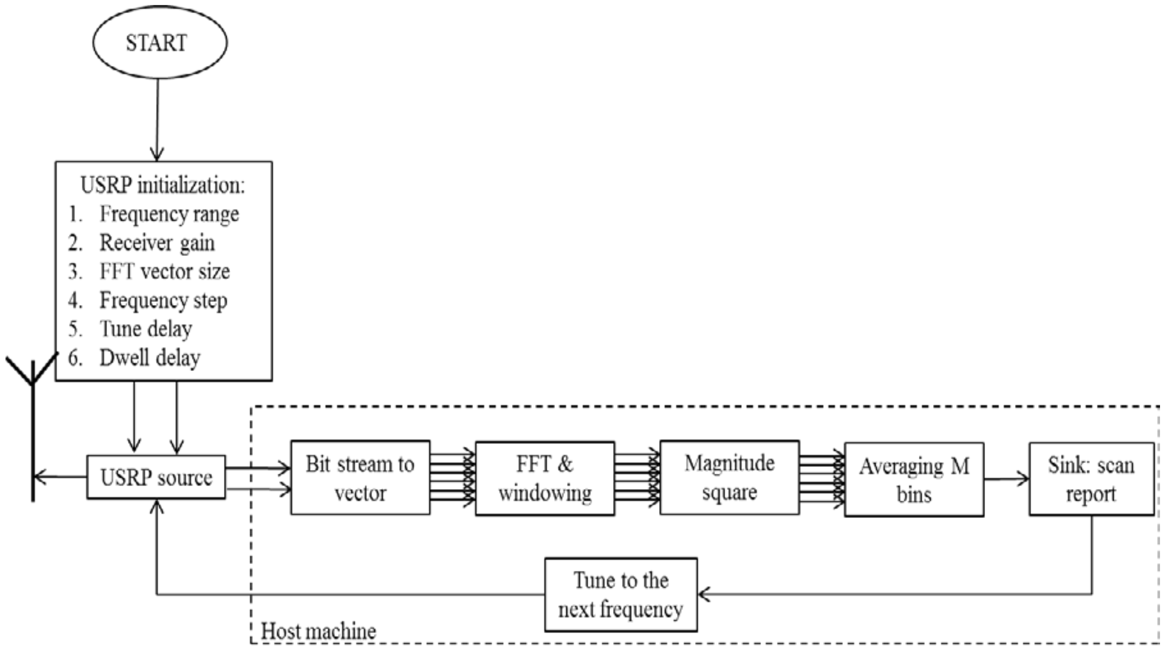

Figure 16. Sensor node flow graph.

a. USRP Initialization

When the script begins, a number of user-selected parameters required by the device to properly run (e.g., sampling rate and vector size, gain) are immediately passed to the USRP after running the script by typing the following line of command in the Ubuntu terminal:

```
Python sensor.py 917000000 918100000 -g 0 -b 500000 -F 1024 -FS 200000
        --tune_delay 0 .1 --dwell_delay 0.05 -thres -58
```

Each of these parameters are explained in the following paragraphs.

The sampling rate *FS* is an important parameter that must be chosen carefully. Care is required because USRPs are programmed to discard any overflowing samples [28], which usually occurs when using a high sampling rate. This in turn causes a variable delay in the scan process in the sensor node. In order to avoid this phenomenon, the sampling rate used was the lowest supported by USRP N210 (200 kHz) [28]. Additionally, in the case of the transmitter, the sampling rate defines the bandwidth of the transmitted signal, which must be less than the channel separation to give non-overlapping channels.

The FFT (fast Fourier transform) size parameter *F* corresponds to the frame size of the FFT block and is exactly the size of the allocated buffer used to accumulate data before any FFT processing, which was arbitrarily set to 1024.

The gain *g* corresponds to the receiver's tuner card gain. By default, the gain is set to 15 dB, which is half of the maximum gain supported by USRP N210. [26] In this experiment, the receiver's gain is set to 0 dB because experimental testing showed that using a gain at the receiver introduces a nonlinear increase in the noise level, which in turn leads to a high probability of false alarm as shown in Table 2 in the next section.

The tune delay *tune_delay* is the wait time necessary for the USRP to tune to the next frequency. All the data collected during this time is discarded. For the overall experiment, the tune delay at sensor nodes is set to 0.1 s [30]. The dwell delay *dwell_delay* is the actual time that the USRP listens to a fixed frequency. Samples taken during this period form the FFT frame, which is set to 0.05 s [30].

The frequency range is the overall spectrum to be scanned. For this experiment, 917 MHz to 918 MHz was chosen because this is a part of the 900 MHz ISM frequency band. The frequency step *b* corresponds to the separation between two adjacent channels in the frequency spectrum and was set to 0.5 MHz, which is greater than the transmitted signal bandwidth so that no overlap between adjacent channels occurs as dictated by the scenario design requirements. The total number of channels was three, which is the number of primary users in this scenario.

The threshold *thres* is selected in order to differentiate between the energy of a sensed signal and the noise level. A further explanation on how the threshold is selected is provided in subsection *c*.

b. Data Flow

After initialization, the USRP starts sending a stream of complex raw (*I* and *Q*) data to the host machine. All received samples during the tune delay are discarded. Samples arriving during the dwell delay are buffered to form a frame of the preselected FFT frame size (F = 1024) using the GNU Radio *bit-stream-to-vector* (*s2v*) block. The frame vector passes through the *FFT* block, which performs a fast Fourier transformation on the signal. A Blackman-Harris window is used to overcome the spectrum leakage effect of the FFT transformation on a non-periodic signal. This functionality is accomplished using the two GNU Radio blocks *window.blackmanharris* and *fft_vcc* as follows [30]:

mywindow = filter.window.blackmanharris(self.fft_size)

ffter = fft.fft_vcc(self.fft_size, True, mywindow, True).

The output vector passes through the *complex_to_mag_squared* (*c2mag*) block, which calculates the squared-magnitude of each bin of the vector. The output is a float type data and is placed in an array defined as *stats* block and labeled *m.data,* whose length corresponds to the FFT size [30]. Finally, all the GNU Radio blocks forming the *sensor.py* flow graph are connected together using the connect function:

self.connect(self.u, s2v, ffter, c2mag, stats).

The first element of the *m.data* array corresponds to the center frequency of the scanned spectrum, while the next 513 elements correspond to the positive side of the channel (frequency domain representation). Conversely, the 514 to 1024 elements correspond to the negative side of channel [30]. As depicted in Figure 17, the USRP introduces a DC offset at the center frequency and slight distortion at both edges due to filtering irregularity. In order to improve accuracy when estimating the signal's power, the first and the last 128 elements are discarded before averaging. The estimated average energy is determined in accordance with

$$E_{av} = 10\log_{10}\left(\frac{\sum_{i=129}^{896} |S[i]|^2}{F-256}\right) \tag{6}$$

where, $|S[i]|$ corresponds to the magnitude of the ith element of the FFT output and F is the frame size of the FFT block.

Next, the result is compared to a preselected threshold (noise level) to decide whether the channel is free or busy, and the result is recorded into the scan report. After completing the entire process, the script returns to the beginning, tunes the USRP to the next channel's center frequency and repeats the process indefinitely.

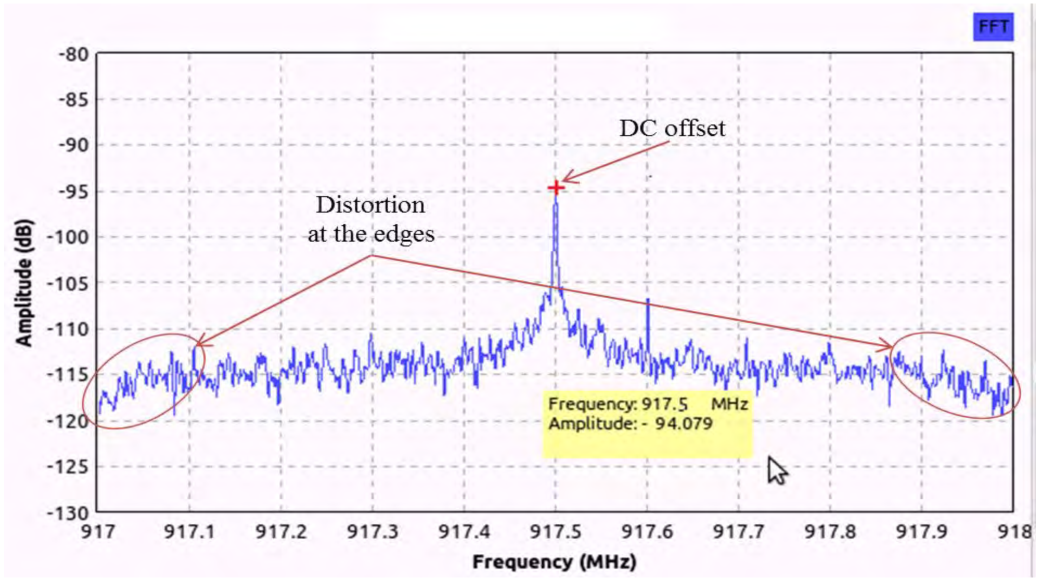

Figure 17. USRP output showing DC offset and edges distortion

c. Noise Level Estimation and Threshold Selection

A python routine similar to the *sensor.py* routine is developed to estimate the noise level of the surrounding environment. The routine loops 1000 times and calculates the average energy for each channel. This intermediate test was carried out twice on the roof of Spanagel Hall at the Naval Postgraduate School, the first time using 0 dB gain at the receiver and the second time using a 15 dB gain. The results are shown in Table 2.

Table 2. Measured noise level for each channel using two receiver gains.

	Channel 1: 917 MHz	Channel 2: 917.5 MHz	Channel 3: 918 MHz
Average noise level (0 dB gain)	−59.530 dBm	−59.697 dBm	−59.331 dBm
Standard deviation (0 dB gain)	0.060	0.039	0.068
Average noise level (15 dB)	−54.693 dBm	−53.231 dBm	−51.465 dBm
Standard deviation (15 dB gain)	44.840	41.456	55.576

The average noise level of the three channels with 0 dB gain is around −59 dBm. Information can be taken from the standard deviation, which is very small for all channels. This means that the noise level for 1000 iterations changes very slightly; however, using a 15 dB gain at the receiver introduces large changes to the noise level. This can be seen from the large standard deviation. This test supports the 0 dB gain choice. In order to choose a correct threshold, another test was carried out. This test was conducted to compute the probability of false alarm P_f corresponding to the number of times the average energy exceeding the preselected threshold N_{HE} divided by the total number of iterations N_T:

$$P_f = \frac{N_{HE}}{N_T}. \tag{7}$$

In the absence of any transmitter, a sensor is set to loop 1,000 times for three different threshold values (−59, −58.5, and −58 dBm). The results are recorded in Table 3. A reasonable choice for the threshold seems to be −58 dBm, with probability of false alarm less than 5%, for two reasons: 1) a selection of a low threshold (high P_f) means that many opportunities may be lost which in turn reduces the efficiency of the system, 2) a selection of a high threshold (low P_f) leads to a possible misdetection of an eventually present primary user with a low signal level, which in turn causes interference between primary and secondary users.

Table 3. Probability of false alarm for each channel with threshold set to −58, −58.5, −59 dBm and no primary is bursting.

	Probability of false alarm P_f at channel 1	Probability of false alarm P_f at channel 2	Probability of false alarm P_f at channel 3
−58 dBm	0.035	0.041	0.029
−58.5 dBm	0.17	0.21	18.5
−59 dBm	0.332	0.297	0.378

d. Cognitive Environment

The proposed scenario calls for an environment in which sensor nodes are deployed around the center of the area, and each node is capable of receiving signals from all of the primary users. To determine where to place each element of the network and verify that the experiment is carried out in the same condition as those of the simulations in [2], a preliminary test was carried out in order to measure the signal power as a function of distance. A primary user was set to send a continuous wave signal with fixed output energy and frequency (i.e., 0.1 W at 917 MHz), while a sensor node measures the received power at different distances from the primary user. The sensor node performs a spectrum sensing scan 1,000 times at each point and averages the received energy

$$E_{rv} = 10\log_{10}\left(\frac{\sum_{i=1}^{1000} E_{av}}{1000}\right) \tag{8}$$

where E_{av} is given by Equation (6).

The results are shown as the received power plotted against distance in Figure 18. The minimum and maximum measured signal powers at each point are also plotted. The results indicate that the minimum signal level for distances larger than 10.0 m is under the preselected threshold (−58 dBm), which causes a possibility of an erroneous detection. To be on the safe side, the maximum distance between primary users, secondary user, and sensor nodes was chosen to be less than 10.0 m.

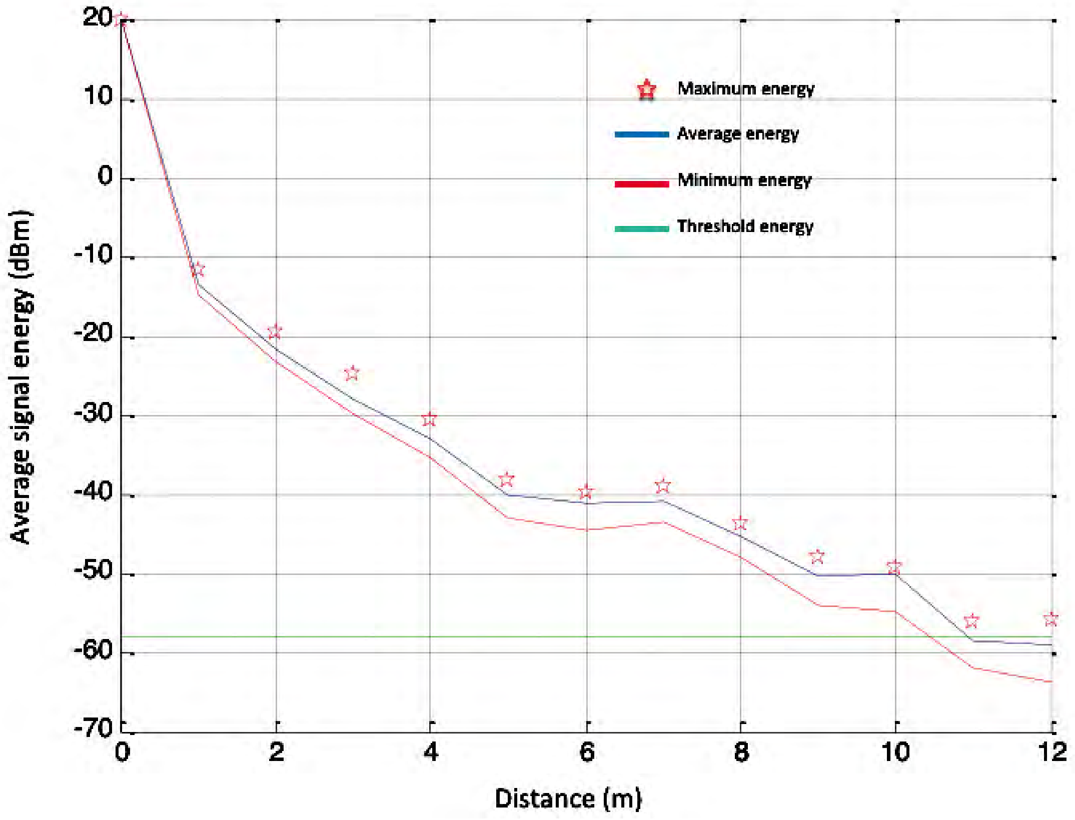

Figure 18. Average signal energy versus distance

3. Secondary User Design

The cognitive radio design was based on the conceptual diagram shown in Figure 10. Essentially, the secondary user should be able to coexist with the primary user in the same frequency spectrum without interference; therefore, the secondary user must perform spectrum sensing before accessing the channel.

The *cognitive.py* routine accomplishes all secondary user functions to meet this end. As shown in Figure 19, the routine starts by initializing the USRP before carrying out an energy-detection process to look for unoccupied channels. Whenever there is an opportunity (i.e., a free channel), the cognitive radio switches into transmission mode. The energy detection process uses almost the same flow graph as the sensor node's routine. That is, after averaging the FFT bins and determining the received signal's average energy, the result is compared to a preselected threshold to decide whether the channel is free or busy. If busy, the algorithm sends a tuning request to the next frequency, and the process is carried out again by calling the function

self.g.set_center_freq(uhd.tune_request(target_freq,rf_freq=(target_freq+self.lo_offset), rf_freq_policy=uhd.tune_request.POLICY_MANUAL)).

This function is responsible for tuning the receiver center frequency to the target frequency, which corresponds to the central frequency of the next channel.

If a free channel is detected, the algorithm switches to transmission mode by calling the developed function *tb.tx_transmitter(center_freq),* which sends a burst following the same process as the one used by a primary user. This happens by generating packets of pseudo-random data, modulating them, and sending them over the air using the USRP. After completing one burst, the secondary user switches back to spectrum sensing mode in the next channel to again look for an unoccupied frequency.

In the first test of the cognitive radio, the same values for tune delay and dwell delay as those used for the sensor nodes (0.1 s and 0.05 s, respectively) are used. This test showed that a leakage of power from the transmitter side to the receiver side occurs even though both sides are physically separated (i.e., using two antennas on two sides, as shown in Figure 20) and the two processes, sensing and transmission, are carried out at two different frequencies.

This leakage leads to very high probability of false alarm in the next channel as shown in Table 4. In order to avoid this phenomenon, an additional separation in the time domain is added. That is, the secondary user introduces a time delay between the transmission process and the follow-on sensing process. This solution has a positive influence on reducing the probability of false alarm, as shown in Table 4. This time domain separation is created by using a larger tuning delay (i.e., 0.2 s) than the one used for the sensor nodes.

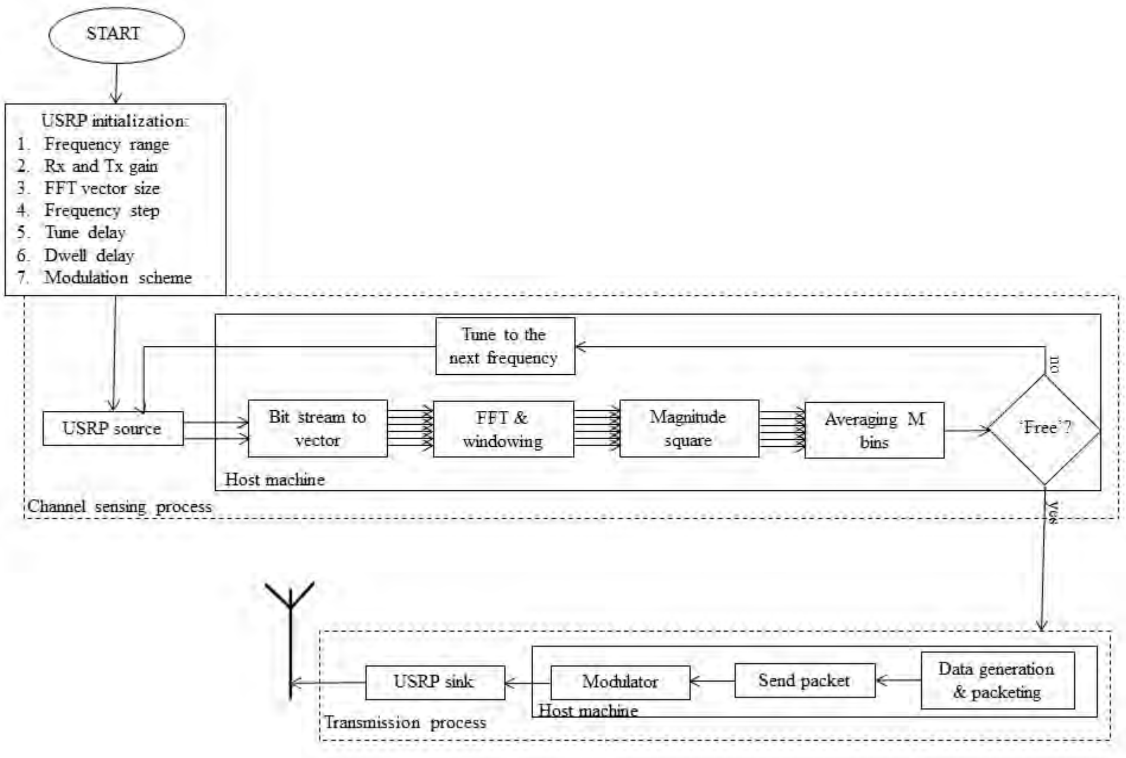

Figure 19. Cognitive radio flow graph.

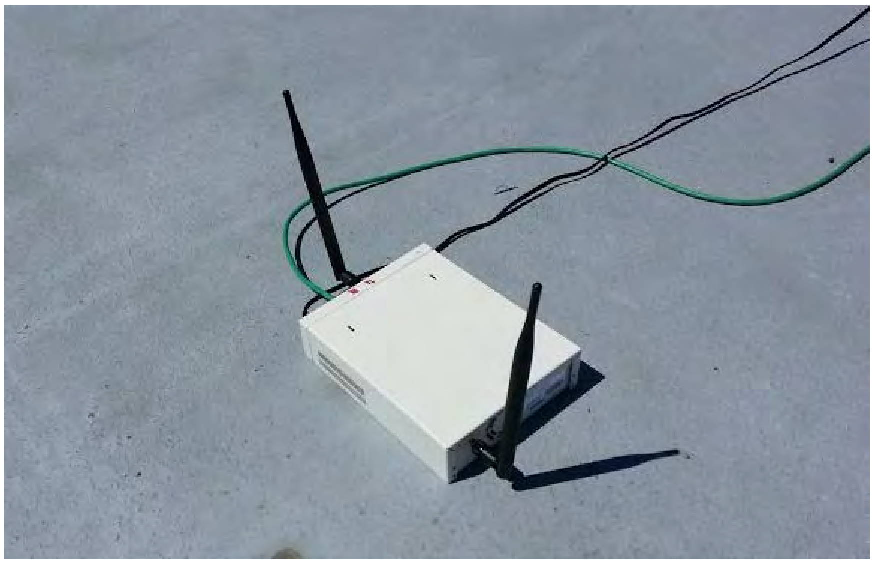

Figure 20. Cognitive radio station using two separated transmitter and receiver antennas.

Table 4. Probability of false alarm versus tune delay for SU.

Tune delay in seconds	P_f at channel 2	P_f at channel 3
0.1	0.447	0.029
0.2	0.049	0.035

C. EXPERIMENTAL RESULTS

1. Testbed

The cognitive environment testbed for the complete scenario was set up on the roof of Spanagel Hall at the Naval Postgraduate School. All network elements were built on the software-defined radio design philosophy using GNU Radio and interfaced with USRPs to take advantage of the software-defined radio's features, which are mainly flexibility and adaptability.

The scenario architecture was first introduced in the discussion of simulation in [2] in which a large number of sensor nodes, primary users and secondary users are deployed in cognitive environment. In this experiment, only four sensors nodes were used to form the wireless sensor network. The four sensor nodes, three primary users, and one secondary user operated in the same proximity to form the overall scenario, as shown in Figure 21.

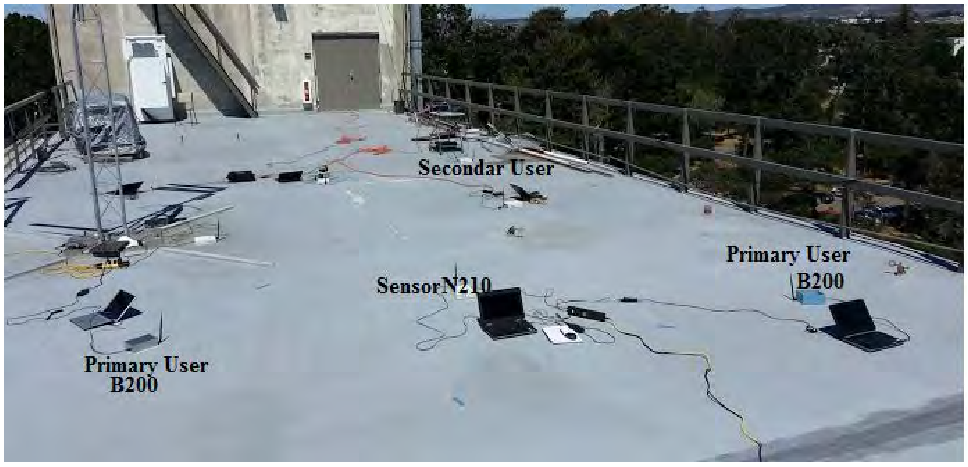

Figure 21. Complete testbed with four sensor nodes, three PUs, and one SU.

A sensor node is composed of a laptop in which the routine *sensor.py* runs interfaced with the USRP N210 with WBX daughterboard. Each sensor node performs a spectrum sensing scan in the band of interest [917–918 MHz].

A secondary user deploys the same USRP device (N210 with WBX daughterboard) as the sensor node and uses a laptop in which a *cognitive.py* routine is executed as explained in the previous section.

The primary users are implemented using USRPs B200. Each one is connected to a laptop via a USB cable. The *primary.py* routine is run so that each primary user transmits a constant amplitude burst in a fixed channel following the concept introduced in the last section. In order to eliminate any possibility of distinguishing between primary and secondary users using measured power levels, the same output power for both is used (0.1 W), even though in the real world the primary user power is greater than secondary user power. A sample of the received energy pattern for a particular channel (channel 3) at each of the sensor nodes is shown in Figure 22. The primary user's traffic can be readily observed in the long continuous bursts dispersed over time. The secondary user traffic is reflected in the short pulses. The average received signal strength of the secondary user signal by each sensor node was −36.652 dBm, −42.232 dBm, −32.819 dBm, −28.561, respectively.

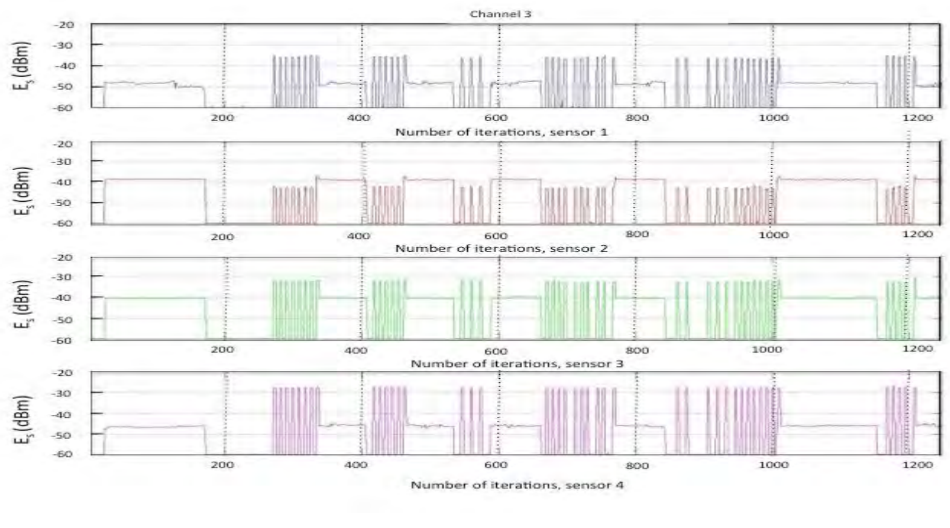

Figure 22. Received energy pattern at each of the sensor nodes in channel 3.

All the elements of the network are assigned stationary positions as listed in Table 5. The positions of all elements (i.e., sensor nodes, primary users, and the secondary user) are scaled by a factor of 100 to better illustrate the results obtained.

Table 5. Primary, sensor nodes and secondary user coordinates used in the testbed of ESRB localization scheme.

Users	X-Coordinates (cm)	Y-Coordinates (cm)
Primary User 1	−800	250
Primary User 2	800	250
Primary User 3	800	−250
Sensor Node 1	0	−200
Sensor Node 2	200	0
Sensor Node 3	0	200
Sensor Node 4	−200	0
Secondary User	−400	250

2. ESRB Localization Scheme Results

Test execution is run for 20 superframes with the WSN performing 55 scans per superframe. The position estimates obtained from the ESRB localization algorithm using the experimental data from the sensor nodes are shown in the Figure 23 by the magenta crosses. The primary users are the green boxes, while the secondary user is the blue circle. The sensor nodes are the red Xs in the center of the plot.

The distance error of the estimated position of the secondary user is plotted against the number of superframes, as shown in Figure 24. The first 18 superframes result in diverged position estimates from the algorithm. These position estimates are the magenta crosses inside the circular ring formed by the wireless sensor network at the center of the environment, as shown in Figure 23. The ESRB algorithm can and will diverge when the iterative non-linear least-squares method is unable to determine the direction of descent towards the local minimum. This can be due to several factors but is primarily due to the algorithm failing to identify the secondary user from among the primary user traffic. At superframe 18, a coarse position estimate is identified at (−2385 cm, 2385 cm), but this is not close to the secondary user position (an error of 2915 cm).

Additionally, a second potential position was discovered later at (−2380 cm, 1440 cm), which is also incorrect (an error of 1978 cm). These coarse position estimates are not accurate but are observed to be concentrated in the vicinity of and converging towards the true position of the secondary user. Those errors are most likely because of the large amount of spectral scans required in order to obtain an accurate estimate of the probability of detection at each sensor node and a lack of timing synchronization among sensors. Also, the simulation results in [2] suggest the number of sensor nodes required to be on the order of 50 compared to four in this work.

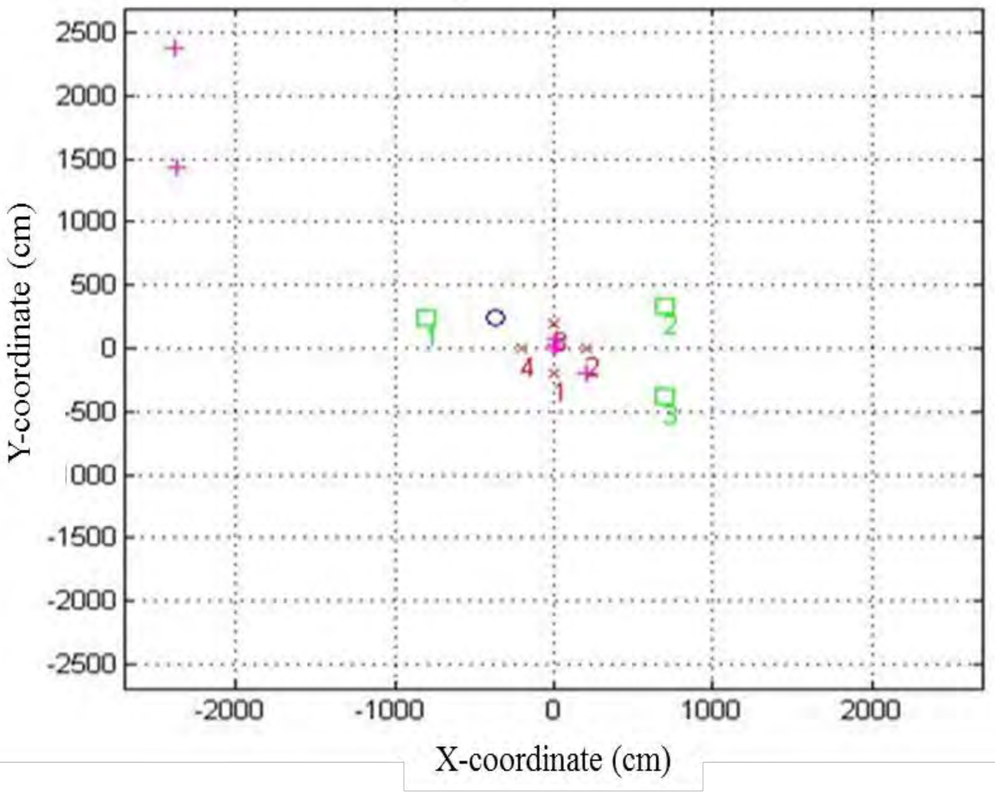

Figure 23. Experimental model and results using wireless sensor network to locate a stationary cognitive radio.

3. CRSSB Localization Scheme Results

As an alternative, the collected data was processed by the received-signal-strength-based localization scheme to determine if it was possible to solve for the position of the secondary user. Several assumptions were introduced to facilitate this operation. Specifically, all secondary user traffic was segregated from the primary user traffic on the basis of signal strength and transmission pattern (burst length).

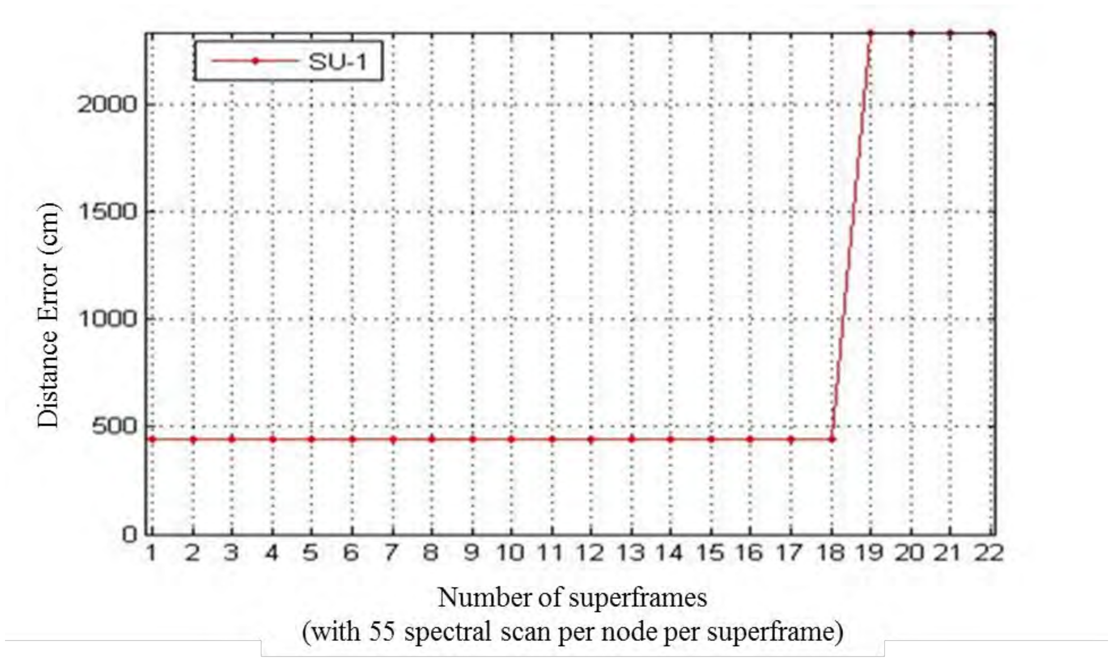

Figure 24. Distance error (cm) versus the number of superframes.

The secondary user's original transmission power was assumed to be 0.1 W. Under these conditions, the following distance estimates were obtained for each of the sensor nodes to form the radii of the circles (sensor 1, 2150.802; sensor 2, 4088.982; sensor 3, 1383.463; sensor 4, 847.302). The lack of intersection of all four circles makes it difficult to achieve a secondary user's position estimate using the received signal strength localization method alone.

As a second alternative, the cooperative-received-signal-strength-based (CRSSB) localization scheme along with the previously mentioned assumptions was used to estimate the secondary user location (under the same conditions as the alternative). The CRSSB did provide an acceptable position estimation; its output corresponds to the area of intersection of three circles as shown in Figure 25. The circle centered on sensor node 2 is discarded since it does not intersect with any other circle (the estimated distance (circle radius) is larger than the other estimations). The resulting estimated position of the secondary user is (−480 cm, 290 cm) with an error of 89 cm.

In this chapter, an in-depth explanation of the adopted testing scenario used to test the ESRB and CRSSB localization schemes was provided. Specifically, the testbed and the network element design were presented in detail. Each of the network elements worked successfully and provided the desired output. Experimental results were presented for both ESRB and CRSSB localization schemes; the CRSSB position estimate was more accurate than the ESRB estimate. The large error in the ESRB estimate is believed to be due to the lack of time synchronization among sensor nodes, the small number of sensor nodes, and a small number of spectral scans per superframe as compared to those used in [2].

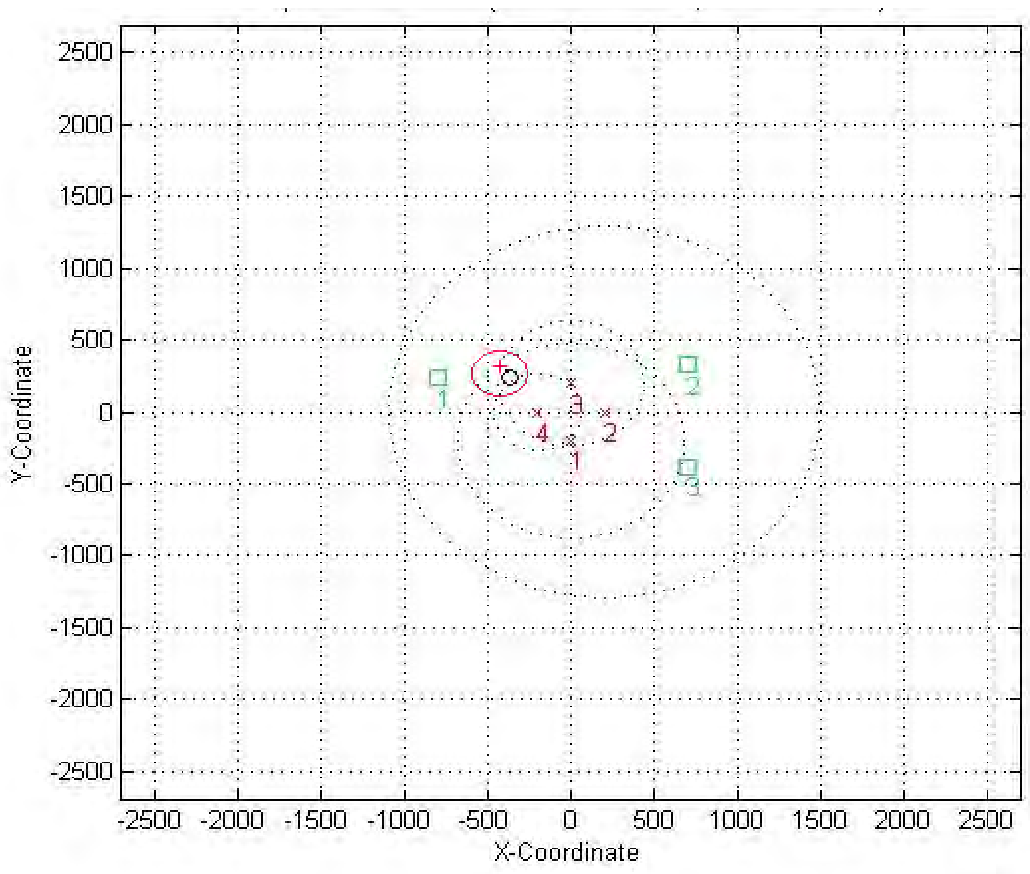

Figure 25. Experimental model and results using received signal strength localization scheme.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

The focus of this thesis was an implementation of a cognitive environment to demonstrate the effectiveness of the ESRB and CRSSB localization schemes under real-world conditions. To accomplish these objectives, a design for the environment was proposed as shown in Figure 7. The cognitive environment is composed of four elements: 1) primary user, 2) secondary user, 3) sensor node, and 4) decision maker. First, an explanation of the approach behind each network element's conceptual diagram was given. Second, the actual implementation of each element using GNU Radio and USRPs was presented in detail. The ESRB localization scheme did not provide accurate position estimates; however, the CRSSB localization scheme yielded an estimate within an acceptable level of tolerance.

A. SIGNIFICANT CONTRIBUTIONS

Three main contributions were presented in this thesis.

To take advantage of software-defined radio features (flexibility and adaptability), all network elements were built using GNU Radio interfaced with the USRPs from Ettus Research. Three GNU Radio routines were developed to meet the design requirements for sensor nodes, primary users and secondary user. Two Ettus products were used to achieve this: the USRP N210 with WBX daughterboard for sensor nodes and the secondary user and the USRP B200 for primary users.

The testbed for the cognitive environment was developed and set up as shown in Figure 21 on the roof of Spanagel Hall at the Naval Postgraduate School. Each of the network elements worked successfully and provided the desired output. The primary users generated a signal with fixed amplitude at the preselected channel. All sensor nodes were then able to perform energy calculations and detect of the primary user's signal. Finally, the secondary user was able to sense the spectrum and transmit a generated burst in the detected vacant slots.

The scan reports from each sensor node were aggregated at the decision maker in which the ESRB and the CRSSB localization algorithms were executed in order to estimate the secondary user location. For the ESRB localization scheme, the results were not accurate but were observed to be concentrated in the vicinity of and converging towards the true position of the secondary user as shown in Figure 23. The errors are believed to be caused by three factors: the limited number of sensor nodes used (four sensor nodes) compared to the number used on the simulation presented in [2] (50 sensor nodes), the number of scans per superframe (55 scans) being less than the suggested number in [2] to obtain accurate estimates (600 scans), and the lack of timing synchronization among sensor nodes. The CRSSB localization scheme provided position estimation within an acceptable level of tolerance.

B. FUTURE WORK

An improvement to the implementation reported in this thesis would be to include a synchronization process among sensor nodes. This is possible via several solutions. For example, the synchronization may be accomplished using a multiple-input/multiple-output (MIMO) cable or using an optional global positioning system (GPS) module, which allows multiple USRPs spread over a large area to synchronize to the GPS standard [26].

The geographic area of the overall cognitive environment implemented in this work was restricted to the roof of a building; however, the actual cognitive network may extend to several kilometers [2]. The main cause of this restriction was the limitation on the output signal power delivered by the USRPs, which was dictated by the capability of the USRP B200 (maximum output signal power is ~ 0.1 W [26]) and the regulations of ISM band, which mandates that the maximum signal power allowed is 0.1 W. Using another frequency band and adding an external amplifier to the device seems to be a reasonable solution to allow the device to reach higher power output so that a wider test area is possible.

In this thesis, the secondary user was assigned a static position; however, a real-world scenario seldom presents such behavior. A more realistic test scenario can be accomplished by using a moving secondary user with random movement at various speeds over time.

The simulation results presented in [1] and [2] showed that using a large number of sensor nodes gives more accurate position estimates. In the testbed scenario in this work, the number of sensor nodes was limited to four. The main cause of this restriction was a limited number of the USRPs available for testing. Using additional sensor nodes may provide more accurate results than these experimental results.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

This appendix includes the GNU Radio code for implementing the primary, the sensor node, and the secondary. The code for *Transmit path* class is also listed.

A. PRIMARY

The *banchmarck_tx.py* script from the GNU Radio example library is modified to obtain the *primary.py* python routine, and satisfy the design requirements.

```
#!/usr/bin/env python
from gnuradio import digital
from gnuradio import gr, uhd
from gnuradio import filter
from gnuradio import analog
from gnuradio import blocks
from gnuradio.eng_notation import num_to_str, str_to_num
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import math
import sys
import time, struct
from transmit_path import transmit_path
import random
from datetime import datetime
class tx_transmitter(gr.top_block):
   def __init__(self,modulator, options):
      gr.top_block.__init__(self)
         # ------------------------------------------------------------------      #
args = modulator.extract_kwargs_from_options(options)
      #symbol_rate = options.bitrate / modulator(**args).bits_per_symbol()
         self.txpath = transmit_path(modulator, options)
      # ------------------------------------------------------------------
# Set up USRP to transmit on  daughterboard

d = uhd.find_devices(uhd.device_addr(options.args))
uhd_type = d[0].get('type')
   # blocks connection
stream_args = uhd.stream_args('fc32', channels=range(1))
self.u = uhd.usrp_sink(device_addr=options.args, stream_args=stream_args)
# Set up USRP system based on type
if(uhd_type == "usrp"):
   self.u.set_subdev_spec("A:0")
# Use the tune requests to tune each channel
   #k=random.randint(0,2)
   tr=options.freq
   print "starting frequency %d" % tr
   r = self.u.set_center_freq(tr)
dev_freq=self.u.get_center_freq()
   print "actual frequency %d " % dev_freq
     self.usrp_rate  = options.samp_rate
   print "target sampling rate %d" % options.samp_rate
self.u.set_samp_rate(self.usrp_rate)
dev_rate = self.u.get_samp_rate()
   print "actual sampling rate %d" % dev_rate
# ----------------------------------------------------------------
if options.gain is None:
   # if no gain was specified, use the mid-point in dB
   g = self.u.get_gain_range()
   options.gain = float(g.start()+g.stop())/2.0
      print "no gain mentioned, mid point is used " % options.gain
else:
            self.u.set_gain(options.gain)
            print "target gain %d" % options.gain
            dev_gain=self.u.get_gain()
```

```
                    print "actual gain %d" % dev_gain

            # Set the subdevice spec
            if(options.spec):
               self.u.set_subdev_spec(options.spec)
            # Set the antenna
            if(options.antenna):
               self.u.set_antenna(options.antenna)
            self.connect(self.txpath, self.u)
        ##--------------------------------------------------------
        def main():
               mods = digital.modulation_utils.type_1_mods()


               parser=OptionParser(option_class=eng_option,
conflict_handler="resolve")
               expert_grp = parser.add_option_group("Expert")
               transmit_path.add_options(parser, expert_grp)
               parser.add_option("-f", "--freq", type="eng_float", default=913000000,
                              help="set frequency to FREQ", metavar="FREQ")
               parser.add_option("-a", "--args", type="string", default="",
                              help="UHD device address args [default=%default]")
               parser.add_option("", "--spec", type="string", default=None,
                                help="Subdevice of UHD device where appropriate")
               parser.add_option("-A", "--antenna", type="string", default= None,
                              help="select Tx Antenna where appropriate")
               parser.add_option("-S", "--samp-rate", type="eng_float", default=200e3,
                              help="set sample rate [default=%default]")
               parser.add_option("-g", "--gain", type="eng_float", default=15,
                              help="set gain in dB (default is midpoint)")
               parser.add_option("-m",          "--modulation",          type="choice",
choices=mods.keys(),
                    default='psk',
                    help="Select modulation from: %s [default=%%default]"
                        % (', '.join(mods.keys()),))
               parser.add_option("-s", "--size", type="eng_float", default=4000,
                                 help="set packet size [default=%default]")
               parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
                                 help="set megabytes to transmit [default=%default]")
               parser.add_option("","--discontinuous",action="store_true", default=False,
                                 help="enable   discontinous   transmission   (bursts   of   5
packets)")

               (options, args) = parser.parse_args ()
            #print options
               def send_pkt(payload='', eof=False):
               return tb.txpath.send_pkt(payload, eof)
            #print "mods[options.modulation]", mods[options.modulation]
               # build the graph
               tb = tx_transmitter(mods[options.modulation], options)
               tb.start() # start flow graph
               # Open a file
               fo = open("log.txt", "a")
               fo.write("primary user log file \n");
               # generate and send packets
               nbytes = int(1e6 * options.megabytes)
               n = 0
               pktno = 0
               pkt_size = int(options.size)
          #### delay time to start in order to synchronise all usrp's to the same starting
point
               vdate=time.time()
               print "vdate", vdate
               bdate=datetime(2014, 7, 7, 15, 21, 10)
               t= time.mktime(bdate.timetuple())
```

```
            print "bdate", t

            while t>vdate:
                    time.sleep(1)
                    print "waiting"
                    vdate=time.time()
            while 1:
                    fo = open("log.txt", "a")
                    k=random.randint(0,3)
                    data = (pkt_size - 2) * chr(pktno & 0xff)
                    payload = struct.pack('!H', pktno & 0xffff) + data
                    send_pkt(payload, eof=False)
                    n += len(payload)
                    fo.write( "packet transmition\n");
                    sys.stderr.write('.')
                    if options.discontinuous and pktno % 10 == 9: # number of packets
in one burst
                            k=random.randint(0,3)
                    vdate=datetime.now().strftime('%H:%M:%S.%f \n')
                            fo.write(vdate);
                            time.sleep(1+k*1)
                            print "sleep"
                    pktno += 1
                    fo.close()
            send_pkt(eof=True)
        if __name__ == '__main__':
          try:
            main()
          except KeyboardInterrupt:
            pass
```

A. SENSOR NODE

The *sensor.py* is based on an available python script in the GNU Radio library entitled *usrp_spectrum_sensing.py*.

```
#!/usr/bin/env python
from gnuradio import gr, eng_notation
from gnuradio import blocks
from gnuradio import audio
from gnuradio import filter
from gnuradio import fft
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import sys
import math
import struct
import threading
from datetime import datetime
import datetime as dt
import time
import calendar
class ThreadClass(threading.Thread):
   def run(self):
      return
class tune(gr.feval_dd):
   """
   This class allows C++ code to callback into python.
   """
   def __init__(self, tb):
      gr.feval_dd.__init__(self)
      self.tb = tb
```

```
    def eval(self, ignore):
        """
        This method is called from blocks.bin_statistics_f when it wants
        to change the center frequency.  This method tunes the front
        end to the new center frequency, and returns the new frequency
        as its result.
        """
        try:
            # We use this try block so that if something goes wrong
            # from here down, at least we'll have a prayer of knowing
            # what went wrong.  Without this, you get a very
            # mysterious:
            #   terminate called after throwing an instance of
            #   'Swig::DirectorMethodException' Aborted
            # message on stderr.  Not exactly helpful ;)
           new_freq = self.tb.set_next_freq()
           # wait until msgq is empty before continuing
            while(self.tb.msgq.full_p()):
                #print "msgq full, holding.."
                time.sleep(0.1)
                return new_freq
        except Exception, e:
            print "tune: Exception: ", e
class parse_msg(object):
    def __init__(self, msg):
        self.center_freq = msg.arg1()
        self.vlen = int(msg.arg2())
        assert(msg.length() == self.vlen * gr.sizeof_float)
        # FIXME consider using NumPy array
        t = msg.to_string()
        self.raw_data = t
        self.data = struct.unpack('%df' % (self.vlen,), t)
```

```
class my_top_block(gr.top_block):
  def __init__(self):
    gr.top_block.__init__(self)
    usage = "usage: %prog [options] min_freq max_freq"
    parser = OptionParser(option_class=eng_option, usage=usage)
    parser.add_option("-a", "--args", type="string", default="",
              help="UHD device device address args [default=%default]")
    parser.add_option("", "--spec", type="string", default=None,
               help="Subdevice of UHD device where appropriate")
    parser.add_option("-A", "--antenna", type="string", default=None,
              help="select Rx Antenna where appropriate")
    parser.add_option("-s", "--samp-rate", type="eng_float", default=1e6,
              help="set sample rate [default=%default]")
    parser.add_option("-g", "--gain", type="eng_float", default=None,
              help="set gain in dB (default is midpoint)")
    parser.add_option("", "--tune-delay", type="eng_float",
              default=0.25, metavar="SECS",
              help="time to delay (in seconds) after changing frequency
[default=%default]")
    parser.add_option("", "--dwell-delay", type="eng_float",
              default=0.25, metavar="SECS",
              help="time to dwell (in seconds) at a given frequency
[default=%default]")
    parser.add_option("-b", "--channel-bandwidth", type="eng_float",
              default=6.25e3, metavar="Hz",
              help="channel bandwidth of fft bins in Hz [default=%default]")
    parser.add_option("-l", "--lo-offset", type="eng_float",
              default=0, metavar="Hz",
              help="lo_offset in Hz [default=%default]")
    parser.add_option("-q", "--squelch-threshold", type="eng_float",
              default=None, metavar="dB",
              help="squelch threshold in dB [default=%default]")
```

```
                parser.add_option("-F", "--fft-size", type="int", default=None,
                              help="specify          number          of          FFT          bins
[default=samp_rate/channel_bw]")
                (options, args) = parser.parse_args()
                   if len(args) != 2:
                   parser.print_help()
                   sys.exit(1)
                self.channel_bandwidth = options.channel_bandwidth
                self.min_freq = eng_notation.str_to_num(args[0])
                self.max_freq = eng_notation.str_to_num(args[1])
                if self.min_freq > self.max_freq:
                  # swap them
                   self.min_freq, self.max_freq = self.max_freq, self.min_freq
               # build graph
                self.u = uhd.usrp_source(device_addr=options.args,
                                   stream_args=uhd.stream_args('fc32'))
               # Set the subdevice spec
                if(options.spec):
                   self.u.set_subdev_spec(options.spec, 0)
                # Set the antenna
                if(options.antenna):
                   self.u.set_antenna(options.antenna, 0)
                self.u.set_samp_rate(options.samp_rate)
                self.usrp_rate = usrp_rate = self.u.get_samp_rate()
                self.lo_offset = options.lo_offset
               if options.fft_size is None:
                   self.fft_size = int(self.usrp_rate/self.channel_bandwidth)
                      print self.fft_size
                else:
                   self.fft_size = options.fft_size # very slow for 128 fft_size
                self.squelch_threshold = options.squelch_threshold
                s2v = blocks.stream_to_vector(gr.sizeof_gr_complex, self.fft_size)
```

```
        mywindow = filter.window.blackmanharris(self.fft_size)
        ffter = fft.fft_vcc(self.fft_size, True, mywindow, True)
        power = 0
        for tap in mywindow:
            power += tap*tap
        c2mag = blocks.complex_to_mag_squared(self.fft_size)
        # Set the freq_step to 75% of the actual data throughput.
        # This allows us to discard the bins on both ends of the spectrum.
        self.freq_step = self.channel_bandwidth
       self.min_center_freq = self.min_freq
        nsteps = math.ceil((self.max_freq - self.min_freq) / self.freq_step)
        self.max_center_freq = self.min_center_freq + (nsteps * self.freq_step)
        self.next_freq = self.min_center_freq
        tune_delay      =    max(0,    int(round(options.tune_delay    *    usrp_rate    /
self.fft_size)))  # in fft_frames
            print "tune delay", tune_delay
        dwell_delay    =    max(1,    int(round(options.dwell_delay    *    usrp_rate    /
self.fft_size))) # in fft_frames
            print " dwell delay", dwell_delay
        self.msgq = gr.msg_queue(1)
        self._tune_callback = tune(self)        # hang on to this to keep it from being
GC'd
        stats = blocks.bin_statistics_f(self.fft_size, self.msgq,
                                self._tune_callback, tune_delay,
                                dwell_delay)
        # FIXME leave out the log10 until we speed it up
            #self.connect(self.u, s2v, ffter, c2mag, log, stats)
            self.connect(self.u, s2v, ffter, c2mag, stats)

        if options.gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.u.get_gain_range()
            options.gain = float(g.start()+g.stop())/2.0
        self.set_gain(options.gain)
        print "gain =", options.gain
    def set_next_freq(self):
        target_freq = self.next_freq
        self.next_freq = self.next_freq + self.freq_step
        if self.next_freq >= self.max_center_freq:
            self.next_freq = self.min_center_freq
        if not self.set_freq(target_freq):
            print "Failed to set frequency to", target_freq
            sys.exit(1)
        return target_freq
    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.
        Args:
            target_freq: frequency in Hz
        @rypte: bool
       """
        r = self.u.set_center_freq(uhd.tune_request(target_freq, rf_freq=(target_freq
+ self.lo_offset),rf_freq_policy=uhd.tune_request.POLICY_MANUAL))
        if r:
            return True
        return False
    def set_gain(self, gain):
        self.u.set_gain(gain)
    def nearest_freq(self, freq, channel_bandwidth):
        freq = round(freq / channel_bandwidth, 0) * channel_bandwidth
        return freq
def main_loop(tb):
    fo = open("sensor1", "w+")
    vdate=datetime.now().strftime('%D')
    #fo.write(vdate);
    fo.write("sensor 1 log file for: %s \n\n\n" % vdate)
    fo.write("0=busy \n");
    fo.write("1=free \n");
    bin_start =int(tb.fft_size * ((1 - 0.75) / 2)) # remove the edges of the signal
    print "bin_start", bin_start
    bin_stop = int(tb.fft_size - bin_start)
    print "bin_stop", bin_stop
    iteration=0
    #### delay time to start in order to synchronise all usrp's to the same starting
point
    vdate=time.time()
    print "vdate", vdate
    bdate=datetime(2014, 7, 7, 14, 44, 10)
    t= time.mktime(bdate.timetuple())
    print "bdate", t
    while t>vdate:
            time.sleep(1)
            print "waiting"
            vdate=time.time()
   ############################################################
    while 1:
    #for j in range (1, 11):
      iteration +=1
      fo = open("sensor1", "a+")
      vdate=datetime.now().strftime('%H:%M:%S.%f \n')
      fo.write(vdate);
      #fo.close()
      print "iteration", iteration, datetime.now()
      for i in range (1,4):
            fo = open("sensor1", "a+")
        # Get the next message sent from the C++ code (blocking call).
        # It contains the center frequency and the mag squared of the fft
          m = parse_msg(tb.msgq.delete_head())
          # m.center_freq is the center frequency at the time of capture
          # m.data are the mag_squared of the fft output
          # m.raw_data is a string that contains the binary floats.
          # You could write this as binary to a file.
            center_freq = m.center_freq
            power_db=0
            power=0
          for i_bin in range(bin_start, bin_stop):
              power +=m.data[i_bin]
            #print power
            power_db += 10*math.log10(1e-6+power/tb.usrp_rate)
          if (power_db > tb.squelch_threshold):
              print "center_freq", center_freq, "power_db", power_db, "channel %d"
%i, "is busy"
                  a=str(power_db)
                  fo.write("chan %d 0 " % int(i));
                  fo.write(a);
                  fo.write("\n");
                  fo.close()
            else:
                  print "center_freq", center_freq, "power_db", power_db, "channel
%d" %i, "is free"
                  a=str(power_db)
                  fo.write("chan %d 1 " % int(i));
              fo.write(a);
                  fo.write("\n");
                  fo.close()
      if __name__ == '__main__':
        t = ThreadClass()
        t.start()
        tb = my_top_block()
```

```
    try:
        tb.start()
        main_loop(tb)
    except KeyboardInterrupt:
        pass
```

B. SECONDARY USER

```
#!/usr/bin/env python
from gnuradio import digital
from gnuradio import gr
from gnuradio import uhd
from gnuradio import filter
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio.eng_notation import num_to_str
from gnuradio.eng_notation import str_to_num
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import math
import sys
import time
import struct
from transmit_path import transmit_path
from gnuradio import audio
from gnuradio import fft
import threading
from datetime import datetime
######################################################################
options={}
frequency=0
```

```
mods = digital.modulation_utils.type_1_mods()
class ThreadClass(threading.Thread):
    def run(self):
        return
class tune(gr.feval_dd):
    """
    This class allows C++ code to callback into python.
    """
    def __init__(self, tb):
        gr.feval_dd.__init__(self)
        self.tb = tb
    def eval(self, ignore):
        """
        This method is called from blocks.bin_statistics_f when it wants
        to change the center frequency.  This method tunes the front
        end to the new center frequency, and returns the new frequency
        as its result.
        """
        try:
            # We use this try block so that if something goes wrong
            # from here down, at least we'll have a prayer of knowing
            # what went wrong.  Without this, you get a very
            # mysterious:
            #   terminate called after throwing an instance of
            #   'Swig::DirectorMethodException' Aborted
            # message on stderr.  Not exactly helpful ;)
            new_freq = self.tb.set_next_freq()
             # wait until msgq is empty before continuing
            while(self.tb.msgq.full_p()):
                #print "msgq full, holding.."
                time.sleep(0.1)
               return new_freq
            except Exception, e:
               print "tune: Exception: ", e
      class parse_msg(object):
         def __init__(self, msg):
            self.center_freq = msg.arg1()
            self.vlen = int(msg.arg2())
            assert(msg.length() == self.vlen * gr.sizeof_float)
            # FIXME consider using NumPy array
            t = msg.to_string()
            self.raw_data = t
            self.data = struct.unpack('%df' % (self.vlen,), t)
      # sensing and main class
      class my_top_block(gr.top_block):
            def __init__(self):
            global options, frequency, mods #modulator
               gr.top_block.__init__(self)
               parser                    =                    OptionParser(option_class=eng_option,
conflict_handler="resolve")
               expert_grp = parser.add_option_group("Expert")
               transmit_path.add_options(parser, expert_grp)
               usage = "usage: %prog [options] min_freq max_freq"
            parser = OptionParser(option_class=eng_option, usage=usage)
            parser.add_option("-a", "--args", type="string", default="",
                        help="UHD device device address args [default=%default]")
            parser.add_option("", "--spec", type="string", default=None,
                           help="Subdevice of UHD device where appropriate")
            parser.add_option("-A", "--antenna", type="string", default=None,
                        help="select Tx/Rx Antenna where appropriate")
            parser.add_option("-S", "--samp-rate", type="eng_float", default=200e3,
                        help="set sample rate [default=%default]")
            parser.add_option("", "--tune-delay", type="eng_float",
                        default=0.25, metavar="SECS",
                  help="time to delay (in seconds) after changing frequency
[default=%default]")
        parser.add_option("", "--dwell-delay", type="eng_float",
                  default=0.25, metavar="SECS",
                  help="time to dwell (in seconds) at a given frequency
[default=%default]")
        parser.add_option("-b", "--channel-bandwidth", type="eng_float",
                  default=500e3, metavar="Hz",
                  help="channel bandwidth of fft bins in Hz [default=%default]")
        parser.add_option("-l", "--lo-offset", type="eng_float",
                  default=0, metavar="Hz",
                  help="lo_offset in Hz [default=%default]")
        parser.add_option("-q", "--squelch-threshold", type="eng_float",
                  default=None, metavar="dB",
                  help="squelch threshold in dB [default=%default]")
        parser.add_option("-F", "--fft-size", type="int", default=None,
                  help="specify number of FFT bins
[default=samp_rate/channel_bw]")
         parser.add_option("-g", "--txgain", type="eng_float", default=15,
                       help="set gain in dB (default is midpoint)")
         parser.add_option("-G", "--rxgain", type="eng_float", default=15,
                       help="set gain in dB (default is midpoint)")
         parser.add_option("-m","--modulation", type="choice",
choices=mods.keys(),
                default='psk',
                help="Select modulation from: %s [default=%%default]"
                  % (', '.join(mods.keys()),))
         parser.add_option("-s", "--size", type="eng_float", default=4000,
                     help="set packet size [default=%default]")
         parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
                     help="set megabytes to transmit [default=%default]")
         (options, args) = parser.parse_args ()
        if len(args) != 2:
          parser.print_help()
          sys.exit(1)
          # Set up USRP to transmit on  daughterboard
          d = uhd.find_devices(uhd.device_addr(options.args))
        uhd_type = d[0].get('type')
          # build graph
          modulator=mods[options.modulation]
        stream_args = uhd.stream_args('fc32', channels=range(1))
        self.u = uhd.usrp_sink(device_addr=options.args, stream_args=stream_args)
          self.g                                                              =
uhd.usrp_source(device_addr=options.args,stream_args=stream_args)
        self.txpath = transmit_path(mods[options.modulation], options)
          ########### transmitter features ########
        # Set the antenna
        if(options.antenna):
          self.u.set_antenna(options.antenna, 0)
          # Set the subdevice spec
        # Set up USRP system based on type
        if(uhd_type == "usrp"):
          self.u.set_subdev_spec("A:0")
          #if(options.spec):
        #self.u.set_subdev_spec("A:0")

        # Set the antenna
        if(options.antenna):
          self.u.set_antenna(options.antenna, 0)
          ###### sampling rate
          self.u.set_samp_rate(options.samp_rate)
          # set gain:
          self.u.set_gain(options.txgain)
          print "target gain %d" % options.txgain
                #dev_gain=self.u.get_gain()
```

```
              #print "actual gain %d" % dev_gain
           # Set up USRP system based on type
        self.channel_bandwidth = options.channel_bandwidth
        self.min_freq = eng_notation.str_to_num(args[0])
        self.max_freq = eng_notation.str_to_num(args[1])
        if self.min_freq > self.max_freq:
           # swap them
           self.min_freq, self.max_freq = self.max_freq, self.min_freq
        # Set the subdevice spec
        #if(options.spec):
        #self.g.set_subdev_spec("A:0")
           #self.g.subdev.set_auto_tr(True)
# Set the antenna
        if(options.antenna):
        self.g.set_antenna(options.antenna, 0)
        self.g.set_samp_rate(options.samp_rate)
        self.usrp_rate = usrp_rate = self.g.get_samp_rate()
        self.lo_offset = options.lo_offset
        if options.fft_size is None:
           self.fft_size = int(self.usrp_rate/self.channel_bandwidth)
              print self.fft_size
        else:
           self.fft_size = options.fft_size # very slow for 128 fft_size
          self.squelch_threshold = options.squelch_threshold
        s2v = blocks.stream_to_vector(gr.sizeof_gr_complex, self.fft_size)
        mywindow = filter.window.blackmanharris(self.fft_size)
        ffter = fft.fft_vcc(self.fft_size, True, mywindow, True)
        power = 0
        for tap in mywindow:
         power += tap*tap
        c2mag = blocks.complex_to_mag_squared(self.fft_size)
        # Set the freq_step to 75% of the actual data throughput.
        # This allows us to discard the bins on both ends of the spectrum.
        self.freq_step = self.channel_bandwidth
          self.min_center_freq = self.min_freq
        nsteps = math.ceil((self.max_freq - self.min_freq) / self.freq_step)
        self.max_center_freq = self.min_center_freq + (nsteps * self.freq_step)
        self.next_freq = self.min_center_freq
        tune_delay  = max(0, int(round(options.tune_delay * usrp_rate /
self.fft_size)))  # in fft_frames
          print "tune delay", tune_delay
        dwell_delay = max(1, int(round(options.dwell_delay * usrp_rate /
self.fft_size))) # in fft_frames
          print " dwell delay", dwell_delay
        self.msgq = gr.msg_queue(1)
        self._tune_callback = tune(self)        # hang on to this to keep it from being
GC'd
        stats = blocks.bin_statistics_f(self.fft_size, self.msgq,
                                self._tune_callback, tune_delay,
                                dwell_delay)
          #self.connect(self.u, s2v, ffter, c2mag, log, stats)




          #### Rx gain
          self.g.set_gain(options.rxgain)
          print "target gain %d" % options.rxgain
          ### connection
          #self.txpath = transmit_path(mods[options.modulation], options)
          self.connect(self.g, s2v, ffter, c2mag, stats)
        self.connect(self.txpath, self.u)
          ##### transmitter
      def tx_transmitter(self, frequency):
          def send_pkt(payload='', eof=False):
          return self.txpath.send_pkt(payload, eof)
          def tx_set_freq(frequency):
                  stream_args = uhd.stream_args('fc32', channels=range(1))
                  l= uhd.usrp_sink("", stream_args=stream_args)
                  r = l.set_center_freq(uhd.tune_request(frequency, rf_freq=(frequency),rf_freq_policy=uhd.tune_request.POLICY_MANUAL))
                  #l.set_center_freq(frequency)
                  print "transmiting on %d frequency " %frequency
          nbytes = int(1e6) #* options.megabytes)
          n = 0
          pktno = 0
          pkt_size = int(options.size)
          tx_set_freq(frequency)
          while 1:
                  data = (pkt_size - 2) * chr(pktno & 0xff)
                  payload = struct.pack('!H', pktno & 0xffff) + data
                  send_pkt(payload, eof=False)
                  #tr.send_pkt(payload, eof=False)
                  n += len(payload)
                  if pktno % 5 == 4:
                          break
                  pktno += 1
            #tr.txpath.send_pkt(eof=True)
          #send_pkt(eof=True
      def set_next_freq(self):
        target_freq = self.next_freq
        self.next_freq = self.next_freq + self.freq_step
        if self.next_freq >= self.max_center_freq:
          self.next_freq = self.min_center_freq
        if not self.set_freq(target_freq):
          print "Failed to set frequency to", target_freq
          sys.exit(1)
        return target_freq
```

```
        def set_freq(self, target_freq):
            """
            Set the center frequency we're interested in.
            Args:
                target_freq: frequency in Hz
            @rypte: bool
            """
            r = self.g.set_center_freq(uhd.tune_request(target_freq, rf_freq=(target_freq
+ self.lo_offset),rf_freq_policy=uhd.tune_request.POLICY_MANUAL))
            if r:
                return True
            return False
        def main_loop(tb):
        global options
        bin_start =250 #int(tb.fft_size * ((1 - 0.75) / 2)) in order to elliminate DC
component
        bin_stop = int(tb.fft_size - bin_start)
        iteration=0
      #### delay time to start in order to synchronise all usrp's to the same starting
point
        vdate=time.time()
        print "vdate", vdate
        bdate=datetime(2014, 7, 7, 15, 25, 10)
        t= time.mktime(bdate.timetuple())
        print "bdate", t
        while t>vdate:
                time.sleep(5)
                print "waiting"
                vdate=time.time()
         while 1:
         iteration +=1
         print "iteration", iteration, datetime.now(),"\n"
         for i in range (1,4):
            # Get the next message sent from the C++ code (blocking call).
            # It contains the center frequency and the mag squared of the fft
            m = parse_msg(tb.msgq.delete_head())
            # m.center_freq is the center frequency at the time of capture
            # m.data are the mag_squared of the fft output
            # m.raw_data is a string that contains the binary floats.
            # You could write this as binary to a file.
              center_freq = m.center_freq
              power_db=0
              power=0
            for i_bin in range(bin_start, bin_stop):
                power +=m.data[i_bin]
            power_db += 10*math.log10(1e-6+power/tb.usrp_rate)
            if (power_db > tb.squelch_threshold):
                print "center_freq", center_freq, "power_db", power_db, "channel %d"
%i, "is busy"
              else:
                    print "center_freq", center_freq, "power_db", power_db, "channel
%d" %i, "is free"
                    tb.tx_transmitter(center_freq)
                    time.sleep(1)
        if __name__ == '__main__':
          t = ThreadClass()
          t.start()
          tb = my_top_block()
          try:
            tb.start()
            main_loop(tb)
          except KeyboardInterrupt:
            pass
```

C. TRANSMIT PATH

```
from gnuradio import gr
from gnuradio import eng_notation
from gnuradio import blocks
from gnuradio import digital
import copy
import sys
class transmit_path(gr.hier_block2):
   #print "transmit_path class"
   def __init__(self, modulator_class, options):
      '''
      See below for what options should hold
      '''
     gr.hier_block2.__init__(self, "transmit_path",
                              gr.io_signature(0,0,0),
gr.io_signature(1,1,gr.sizeof_gr_complex))
self._tx_amplitude = .25 #options.tx_amplitude    # digital amplitude
sent to USRP
      self._modulator_class = modulator_class     # the modulator_class
we are using
      # Get mod_kwargs

mod_kwargs=self._modulator_class.extract_kwargs_from_options(options)
      # transmitter
     self.modulator = self._modulator_class(**mod_kwargs)
     self.packet_transmitter = \
         digital.mod_pkts(self.modulator,
                    access_code=None,
                    msgq_limit=100,
                    pad_for_usrp=True)
      self.amp = blocks.multiply_const_cc(1)
      self.set_tx_amplitude(self._tx_amplitude)
```

```
        # Display some information about the setup
        #if self._verbose:
            #self._print_verbage()
        # Connect components in the flowgraph
        self.connect(self.packet_transmitter, self.amp, self)
    def set_tx_amplitude(self, ampl):
      #print "transmit path set tx amplitude"
        """
        Sets the transmit amplitude sent to the USRP in volts
        Args:
            : ampl 0 <= ampl < 1.
        """
        self._tx_amplitude = max(0.0, min(ampl, 1))
      print self._tx_amplitude
        self.amp.set_k(self._tx_amplitude)

    def send_pkt(self, payload='', eof=False):
      #print "transmit_path send_pkt"
        """
        Calls the transmitter method to send a packet
        """
        return self.packet_transmitter.send_pkt(payload, eof)
    def samples_per_symbol(self):
        return self.modulator._samples_per_symbol
    def differential(self):
        return self.modulator._differential
    def add_options(normal, expert):
        """
        Adds transmitter-specific options to the Options Parser
        """
        if not normal.has_option('--bitrate'):
            normal.add_option("-r", "--bitrate", type="eng_float",
                                        default=100e3,
                                        help="specify bitrate [default=%default].")
                    expert.add_option("-S", "--samples-per-symbol", type="float",
                                    default=2,
                                    help="set samples/symbol [default=%default]")
                    expert.add_option("", "--log", action="store_true",
                                    default=False,
                                    help="Log all parts of flow graph to file (CAUTION:
lots of data)")
                # Make a static method to call before instantiation
  add_options = staticmethod(add_options)
```

LIST OF REFERENCES

[1] A. Adams, M. Tummala, J. McEachen and J. Scrofani, "Source localization and tracking in a cognitive radio environment consisting of frequency and spatial mobility," in *Proc. 7th Int. Conf. on Signal Process. Commun. Syst.,* Carrara, VIC, pp. 1–6, 2013.

[2] A. Adams, "Source localization in a cognitive radio environment consisting of frequency and spatial mobility," Master's thesis, Naval Postgraduate School, Monterey, CA, 2012.

[3] T. Yucek and H. Arsan. "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Survey & Tutorials*, vol. 11, no. 1, pp. 116–130, 2009.

[4] R.A. Rashid, M.A. Sarijari, N. Fisal, S.K.S. Yusof and N.H. Mahalin, "Spectrum sensing measurement using GNU Radio and USRP software radio platform," in *Proc. 7th Int. Conf. Wireless and Mobile Commun*., Luxembourg, 2011.

[5] R.A. Rashid, M.A. Sarijari, N. Fisal, S.K.S. Yusof and S.H.S. Ariffin, "Enabling dynamic spectrum access for cognitve radio using software defined radio platform," in *Proc.of Wireless Technol. Applicat*., Langkawi, pp.180–185, 2011.

[6] C. Carlos, K. Challapali, D. Birru and S. Shankar,. "IEEE 802.22: An introduction to the first wireless standard based on cognitive radio," *J. Commun*., vol. 1, no. 1, pp. 39–47, 2006.

[7] A. He, K.K. Bae, T.R. Newman and J. Gaeddert, "A survey of artificial intelligence for cognitive radios," *IEEE Trans. on Veh. Technol*., vol. 59, no. 4, pp. 1578–1592, 2010.

[8] H. Simon, "Cognitive radio: Brain-empowered wireless communications," *IEEE J. Selected Areas in Commun*., vol. 23, no. 2, pp. 201–220, 2005.

[9] H. Celebi and H. Arslan, "Cognitive positioning systems," *IEEE Trans. on Wireless Commun*., vol. 6, no. 12, pp. 4475–4483, 2007.

[10] V. Sönmezer, "Cooperative wideband spectrum sensing and localization using radio frequency sensor networks," Master's thesis, Naval Postgraduate School, Monterey, CA, 2009.

[11] "Draft standard for wireless regional area network part 22: Cognitive wireless RAN medium access control (MAC) and physical layer (PHY) specification: Policies and procedures for operation in the TV bands," IEEE, 2008.

[12] K.G. Shin, K. Hyooil, A.W. Min and A. Kumar, "Cognitive radio for dynamic spectrum access: From concept to reality," *IEEE Wireless Commun*., vol. 17, no. 6, pp. 64–74, 2010.

[13] J. H. Reed. *Software Radio: A Modern Approach to Radio Engineering*. Upper Saddle River, NJ: Prentice Hall Professional, 2002.

[14] A. Jain, V. Sharma and B. Amrutur, "Soft real time implementation of a cognitve radio testbed for frequency hopping primary satisfying QoS Requirements," in *Proc. 20th Nat. Conf. Commun*., Kanpur, 2014.

[15] E.R. Lavudiya, K.D. Kulat and J.D. Kene, "Implementation and analysis of cognitive radio system using Matlab," *Int. J. Comput. Sci. Telecommun*., vol. 4, no. 7, pp. 23–28, 2013.

[16] D. Cabric, S.M. Mishra and R.W. Brodersen, "Implementation issues in spectrum sensing for cognitive radios," in *Conf. Record 38th Asilomar Conf. Signals, Syst. Comput.,* vol. 1, pp. 772–776, 2004.

[17] "Notice of proposed rule making and order: Facilitating opportunities for flexible, efficient, and reliable spectrum use employing cognitive radio technologies," FCC, 2005.

[18] G. Nautiyal and R. Kumar, "Spectrum sensing in cognitve radio using MATLAB," *Int. J. Eng. Adv. Technol*., vol. 2, no. 5, 2013.

[19] P.S. Aparna and M. Jayasheela, "Cyclostationary feature detection in cognitive radio using different modulation schemes," *Int. J. Comput. Applicat*., vol. 47, no. 21, 2012.

[20] M.A. Sarijari, A. Marwanto, N. Fisal, S.K.S. Yusof and R.A Rashid, "Energy detection sensing based on GNU Radio and USRP: An analysis study," in *Proc. Malaysia Int. Conf. Commun*. (MICC), Kuala Lumpur, 2009.

[21] K. Letaief and W. Zhang, "Cooperative communications for cognitive radio networks," in *Proc. of IEEE*, vol. 97, no. 5, pp. 878–893, 2009.

[22] G. Gnesan and Y. Li, "Cooperative spectrum sensing cognitive radio networks," in *Proc. IEEE Int. Conf. .Commun*., Beijing, 2008.

[23] Z. Wang, Z. Feng, J. Song, Y. Hu and P. Zhang, “A practical semi range-based localizationa for cognitive radio,” in *Proc. IEEE 71st Veh.Technol.* Conf., Taipei, 2010.

[24] M. Zhioyao, W. Chen, K.B. Letaief and Z. Cao., “A semi range-based iterative localization algorithm for cognitive radio networks,” in *Proc. IEEE Trans. Veh. Technol*., vol. 59, no. 2, pp. 704–717, 2010.

[25] I. Guvenc and C. C. Chong, “A survey on TOA based wireless localization and NLOS mitigation techniques,” in *IEEE Communication Survey & Tutorials,* vol. 11, no. 3, pp. 107–124, 2009.

[26] Y. Wang, X. Wang, D. Wang and D.P. Agrawal, “Range-free localization using expected hop progress in wireless sensor networks,” in *IEEE Trans. Parallel and Dist. Syst.*, vol. 20, no. 10, pp. 1540–1552, 2009.

[27] H. Celebi and H. Arslan, “Cognitive positioning systems,” in *IEEE Trans. Wireless Commun*., vol. 6, no. 12, pp. 4475–4483, 2007.

[28] Ettus Research, Product description: USRP N200/N210 Networked series and USRP B200/B210 Bus series. [Online]. Accessed July 2014. Available:https://www.ettus.com/product/details

[29] GNU Radio, “GNU Radio: The free and open radio system” (definition and installation). [Online]. Accessed July 2014. Available:http://gnuradio.org/redmine/projects/gnuradio/wiki

[30] A. Aftab and M. N. Mufti, “Spectrum sensing through implementation of USRP2,” Master’s thesis, Blekinge Institute of Technology, Sweden, 2010.

[31] Python Software Foundation, VA. “Python 3.4.1 documentation.” [Online]. Last updated Sep. 11, 2014. Accessed July 2014. Available: https://docs.python.org/3/

[32] A. Crohas, “Practical implementation of a cognitive radio system for dynamic spectrum access,” Master’s thesis, University of Notre Dame, Indiana, 2008.

[33] A.A. Tabassam, M.U. Suleman, S. Kalsait and S. Khan, “Building cognitve radio in MATLAB Simulink-A Step towards future wireless technologgy,” *Wireless Advanced* (WiAd), pp. 15–20, London, 2011.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California